Develop Applications that Give Users
Decision-Making Expert Advise

Exsys Corvid® Core for Apple Macintosh Developer Manual

Exsys Corvid Core Knowledge Automation Expert System Development Manual

Exsys Inc.
6301 Indian School Rd. NE
Suite 700
Albuquerque, NM 87110
U.S.A.

www.exsys.com

Exsys Corvid Core Manual

1 Overview

Important: Using this Manual

This manual is designed to be used with the online Exsys Corvid Core Tutorials.

(http://www.exsys.com/CorvidCore/tutorials)

When first learning Exsys Corvid, start by watching the **Quick Start** tutorial (part 1 and 2). These quickly show the main features of Corvid and set the context for the other tutorials which provide more detailed step by step instruction on using Corvid and building systems.

Each tutorial has an associated chapter in the manual. The manual chapter generally provides all the information in the tutorial, plus in some cases more detailed specifics than in the tutorials. When such details are only found in the manual, the tutorial will mention this. The tutorials show the actual operation and screens of Corvid.

What is Exsys Corvid Core and what are expert systems?

Exsys Corvid Core is a tool for building and deploying online expert systems.

Expert systems are online programs that help end users solve decision making tasks by interacting with them in a way that emulates the conversation they would have with a human expert to get the advice and assistance they need. Corvid systems do this online via the end user's standard web browser. Expert systems ask a series of precise, focused, relevant questions based on what the end users input and the system logic, to produce precise situation specific advice.

Expert systems automate and deliver the knowledge and approach used by the human expert. This is very different from online search techniques that rely on the user knowing what to ask, and then having to read through the many "hits" it produces to sort out the ones that are authoritative, relevant, up-to-date and complete. Even then, there may be something special about their situation that makes the information they have found incorrect.

Search is great when you are doing general research or are looking for a specific fact, but complex decision making tasks often require expert knowledge that may have taken the expert years to acquire. In practice, when someone needs to know "How do I fix this?", "Does this regulation apply to me?" "What is the best product for me to buy?", "What should I do?", they would really prefer be able to talk to someone that they know is an expert and get their advice. But experts are generally scarce, expensive resources that are not always available.

No one can be an expert in all the areas needed to function everyday. Too many complex things break, too many regulation and procedures need to be followed and in many cases there are just too many options.

Expert systems can provide the ideal alternative in many cases. Expert systems can capture human experts knowledge and make it available online in a way that emulates the conversation with the human expert that users want, to get the answers they need. Instead of having to come up with search queries, users are asked questions that they can answer. Based on their answers, that system will skip unnecessary questions, but focus in on questions that drill down in the relevant areas to produce situation specific advice on precisely what to do. Sessions follow the same series of questions and logic that the human expert would because they are driven by the actual logic and process of the human expert.

What Exsys Corvid Core and Expert Systems are NOT

The description of what expert systems can do often makes people jump to the Hollywood image of artificial intelligence and computers that can just answer any question on their own. That exists ONLY in the movies. Computers that "think" in any real way, do not exist and will not for many years. Even the best natural language interfaces like Siri only really handle fairly simple queries and that can be answered by a quick search.

Exsys Corvid does not "know" how to solve problems out of the box. It is a development tool that allows YOU to describe the logic and process that a human expert uses to solve a problem, and then creates the files and online programs that drive an interactive online session using this logic to interact with the end user to emulate the conversation they would have with the expert.

Heuristic Rules and the Exsys Inference Engine

Corvid uses IF/THEN logic to describe the steps in a decision making process. Most computer users have run across IF/THEN logic either in a programming language or an application like a spreadsheet that does something based on a test expression.

The rules in Corvid, while superficially similar to IF/THEN statements in programming languages are fundamentally very different. In a traditional programming language, if one IF/THEN test is dependent on another IF/THEN test, they must be explicitly linked together in the code - either by nesting the tests or by having code to makes sure they are executed in the correct order. As the number of tests (rules) gets large and there are many interrelationships, hard coding these explicit links becomes very difficult to understand, code and maintain. Small changes in one rule can have major (and sometimes undesired) changes in other sections. Add in probabilistic factors, and it gets even worse. This is why directly hard coding anything more complex than fairly simple logic becomes extremely difficult and expensive to maintain, and hard coding real-world systems that incorporate the many factors experts use to make complex decisions is rarely practical.

Corvid uses a very different approach with heuristic rules processed by an Inference Engine. "Heuristic rules" are often simply defined as "rules of thumb". They are the individual rules the human expert uses to make a decision. A simple way to get the heuristic rules of a decision is to ask an expert that has provided some advice, "How did you come to that conclusion?" They will probably respond with something like "Since I saw X and Y, I knew Z". For example, "Since the temperature was too high and the warning light was blinking, I know it is usually a problem with the controller". This may be a high level rule, and you might ask about the details. For exampled, "Well, how did you determine the temperature was too high?" This would lead to another heuristic rule on that detail, such as "We were working on material A and the temperature should be less than 150 degrees. But for material B, it would be OK up to 180 degrees."

Each of these statements, while not structured in an IF/THEN form can be converted to an IF/THEN rule in Corvid:

IF:

Temperature is too high

and: Warning light is blinking

THEN:

High likelihood that the problem is the controller

IF:

Material being processed is material A

and: [Temperature] >= 150

THEN:

Temperature is too high

IF:

Material being processed is material B

and: [Temperature] >= 180

THEN:

Temperature is too high

These rules contain the same knowledge as the expert's statements, but are now in a form that is both easy for humans to read and understand, and can also be processed and used by the Exsys Corvid Inference Engine to automate the decision.

The Exsys Corvid Inference Engine is able to work with "human readable" rules and use them in an effective way. Instead of having to convert the logic into computer code, rules can be written in English and algebra. Instead of having to hard code explicit linkages between rules, the Inference Engine will simply use the rules as needed to solve a specific problem, using individual rules when, and if, they become relevant.

If the Inference Engine has been told to determine the cause of the problem, it will find the rule that could indicate the problem is the controller. Just having this rule be relevant to determining the cause of the problem, is enough for the Inference Engine to "know" to use it and test to see if the rule can be used. Testing the IF conditions in the rule requires determining if the "Temperature is too high". Instead of having to explicitly hard code links to the rules that can provide this information, the Inference Engine automatically looks through the rules to find any that might be relevant, and finds the rules that would allow it to derive that the "Temperature is too high".

It will now test those rules and get information that can then be used in the higher level rules that were initially tested. This can be repeated as many levels deep as needed. The best part is that all the linkages between rules are dynamic and implicit. There is no need to hard code links between rules. The Inference Engine automatically will use the appropriate rules when, and if, they are needed. Rules can be anywhere in a system and structured however the expert prefers. If an individual rule changes, it can be modified and the effects will automatically be applied anywhere that rule was relevant without having to modify deeply needed IF statements in a hard coded system.

Instead of converting decision making steps into code that only makes sense to a programmer, the human expert can build easily read, understandable rules. Each rule is human readable, but still can be used by the computer.

Corvid is designed to allow non-programmers to build systems that incorporate complex logic. Systems are built by simply describing the various IF/THEN heuristic rules that are used in making in decision. These rules are written in a simple English (or whatever language you prefer) and algebra syntax that is easy to read and understand. In most systems, the order of the rules does not even matter as long as they are all there.

Building a system in Corvid is much more like explaining the steps in the decision making task to another person than programming. Generally the best way to start building a system is to think "how would I teach someone to make this decision"

The Corvid rules are built in Logic Blocks which make it easy to build structured sets of rules that cover all the possible decision-making cases. The Exsys Inference Engine will then use the rules to determine:

- Which rules are applicable at a particular stage of the decision making task.
- What information is needed to determine if those rules are “True” and can be used.
- If the information needed is already available, and if not, how to obtain it.
  - If other rules allow the information to be derived, those rules are used.
  - If the information cannot be derived, the end user will be asked for the information.
- Continuing until all relevant rules have been used, and then displaying the results.

All the Inference Engine needs is the set of rules that are applicable to the decision and commands to tell what task to perform.

Since the inference uses the rules dynamically based on what it is trying to do at a particular step, the order of the rules generally does not matter. Also, since a rule will automatically be used to provide the information needed by a higher level rule, the system automatically can use “subsets” of rules to derive information used in higher level more general rules. This is called Backward Chaining and makes building complex systems much easier.

When the end user needs to be ask to provide input, it is because a relevant rule needs that information and it cannot be derived from other rules. If the information is not needed, it will not be asked - no irrelevant questions. But, all relevant rules will be used, so all logic that might apply to the decision will always be used - infrequently used, special cases will not be overlooked.

To ask the user for input, the Inference Engine displays a question in their browser window. The developer can format how questions will be asked to match any site look and feel. Corvid includes both Java applet and Java servlet versions of the Exsys Inference Engine as Runtime programs. The Applet Runtime runs on the end users machine as a Java applet in their browser window making it very easy to field systems online. The more flexible servlet version runs on a server, with question and result screens designed using HTML forms. Since the servlet only sends HTML to the end user, it allows systems to be run on iPhones and iPads that do not support applets, as well as providing more design options and better compatibility with all browsers.

Both the applet and servlet Inference Engine runtime programs use the powerful and proven Exsys Inference Engine.

Who is Exsys Inc?

Exsys Inc has been in the expert system development tool business since 1983. For over 30 years, Exsys tools have been used to build tens of thousands of expert systems in business, industry, government and the military. Exsys systems have been built for a vast range of complex problems including diagnosing jet aircraft, defending naval ships, keeping Fortune 100 companies productive, and implementing complex regulations. A quick look at some of the case studies on the Exsys web site shows the wide range of problems Exsys systems have solved (Go to http://www.exsys.com and select “Case Studies” under “Sample Systems”) - and those are only a small sample.

Exsys has focused on designing tools that are easy to use and practical. Exsys tools make it easy for developers to build and field systems that can handle very complex decision making tasks. Exsys has 30 years experience in working with companies on what is really needed to describe complex logic in way that is easy for non-programmers to learn. This very pragmatic approach to building expert systems, that has proven very effective. The Java based Inference Engine runtime programs are extensively proven over a wide range of problem types.

For many years, the Exsys's development tool has been the full Exsys Corvid program. It currently runs only in MS Windows and at over $10,000 is an expensive tool primarily limited to large companies, government and military projects. Despite its price, Exsys Corvid is very popular, with Corvid systems typically producing ROIs of 10 to 1, and sometimes even 100 to 1.

Exsys Corvid Core vs Exsys Corvid

Exsys Corvid **Core** is a functional subset of Exsys Corvid. As the name suggests "Core" is the core functionality of Exsys Corvid. "Core" does not support some of the more advanced expert system features of the full Exsys Corvid, but provides the functionality that is more than enough to handle many expert system problems. Exsys Corvid Core uses the same proven, powerful Exsys Inference Engine and development methodology as the full Exsys Corvid, with variables, Logic Blocks and a Command Block, but has a new completely redesigned development environment.

Exsys Corvid Core was designed from scratch for the Mac with an OSX style approach using new single window interface. The underlying capabilities of OSX made it possible to implement many new development features not found even in full Exsys Corvid.

Corvid Core is also tightly integrated with the Exsys Servlet Runtime and Apache Tomcat, making it far easier to build and test systems using the servlet version of the Inference Engine.

So why is Exsys Corvid Core so much lower priced than full Exsys Corvid? The full Exsys Corvid is a very powerful development tool, and for complex, advanced expert systems, it is still the tool to use. But many systems don't need all the power of Exsys Corvid - and realistically, many groups don't have the budget for it. The web is how everyone distributes knowledge, and the vast majority of information oriented web sites are struggling to provide help in ways that could be done much better with expert systems. Exsys Corvid Core provides far more than enough power for many decision-making problems at a low cost. Over 30 years ago, Exsys's first expert system development tools cost only a few hundred dollars. Their price went up with the tool's power and with a focus on a specialized markets. Exsys Corvid Core returns to that low price point since the web now makes expert system development and deployment the ideal solution for many web sites.

Building an Exsys Corvid Core System

Corvid is designed to be easy to use. Rules are just If/Then statements in English and algebra like you would explain them to another person. There can be high level rules that describe high level logic, and other rules that fill in the details about the factors used in the high level rules.

Corvid uses Logic Blocks to organize related rules in tree structured diagrams. The diagrams are only to help developers organize and fill in all the gaps in the rules. Corvid does not care how the Logic Blocks are structured, provided all the needed rules are created. A system may have one tree or many, and the trees are not traditional "flow chart" style trees since in a probabilistic system, you may be 70% on one branch and 30% on another parallel branch. If there are existing tree or process flow diagrams, they can usually be directly converted into Corvid Logic Blocks.

Logic Blocks are also a great way to approach problems where the expert has never fully documented how the make a decision. Adding a few nodes to a tree highlights other possible paths and helps the expert fill in what they would do in various situations. This both builds an expert system, and captures and documents the expert's knowledge in precise detail.

There can be subsets of rules to derive specific lower level facts. Those rules will automatically be used when, and if, they are needed. All linkages between rules are implicit based on what the Inference Engine determines is needed. If the lower level rule logic changes, just change the rules and the changes will automatically carry into the higher level logic.

A Command Block provide the procedural control of how a system runs. These are often just 2 or 3 commands to start backward chaining and display results, but can be much more extensive when a system requires it.

Systems can be run and tested at any time with the click of the mouse. They can be run with either the applet or servlet versions of the runtime Inference Engine. Running with the servlet requires Apache Tomcat, but that is a free download from Oracle and Corvid helps with the install and takes care of the housekeeping issues of running with Tomcat.

The end user interface can be designed by setting properties for layout, fonts, colors, images, etc. The Applet Runtime runs in a region on an HTML page, and everything outside of the applet window can be designed with normal HTML code. The servlet runtime uses HTML forms based on templates to ask questions or display results. Corvid includes default HTML templates that can be edited with any HTML editor to match a site's look and feel. Deploying with the Applet Runtime is as easy a putting the files Corvid generates on your standard web server. Deploying a system on the web with the servlet runtime requires a server with Apache Tomcat, Glassfish, IBM Websphere or other support for Java servlets, but again Corvid builds all the files you need.

2 Configuring Your Mac for Corvid

Note: This is important and should be done before using Corvid.

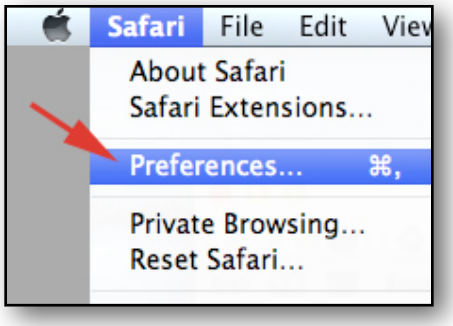
Systems can be created using the applet or servlet runtime, but the applet approach provides a more extensive trace feature and is generally easier to start with. Once the system logic is working correctly in the applet mode, it is easy to switch over to the servlet for user interface design and fielding.

The Safari browser should be set to allow systems to run with the Corvid Applet Runtime. This is a simple configuration option for Safari (which you may already have set if you do web development) Using the Corvid Servlet Runtime requires installing Apache Tomcat, but this is a free download from Oracle and easy to install. (Installing Tomcat is covered in chapter 14)

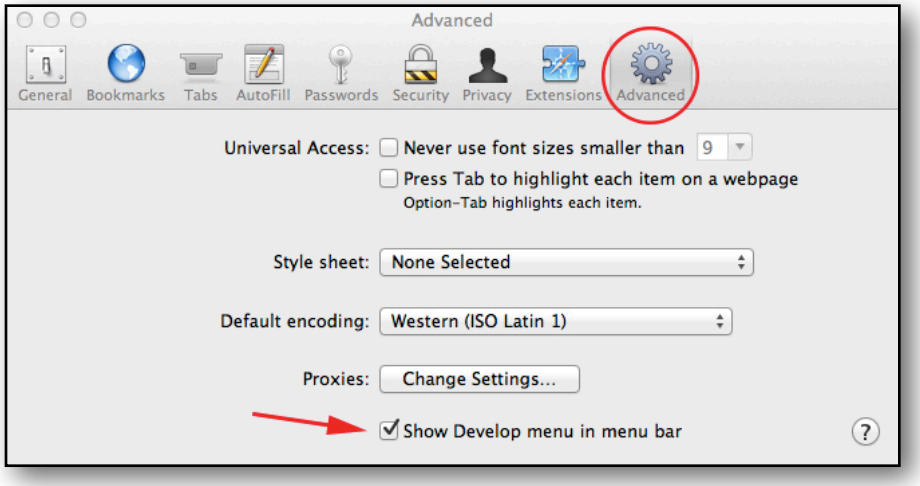
2.1 Configuring Safari for the Applet Runtime

Developing Corvid systems with the Corvid Applet Runtime requires that Safari have a special option set that allows it to run Java Applets locally. This applies ONLY to the development machine since there is no configuration requirement needed to run Java Applets from sites on the web. When the system is fielded, your end users do NOT need to make this change. (This seems like an odd behavior since local applets are far more trustworthy than applets over the web, but it is the way Safari works at the moment. Hopefully this will change in the future.)

1. Open Safari and select “Preferences”.

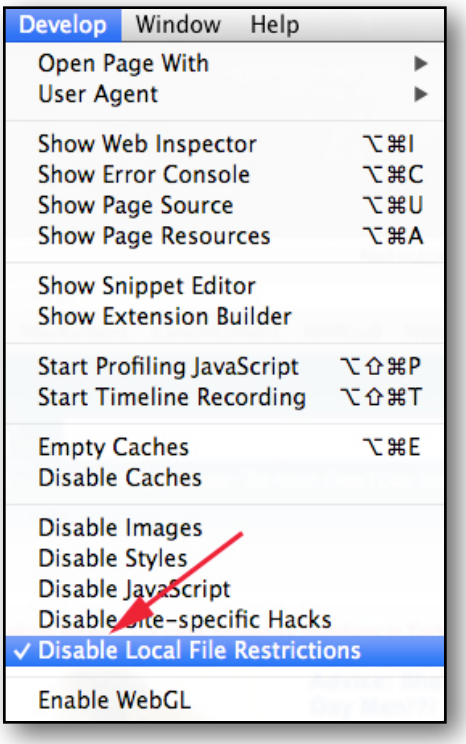


2. Go to the “Advanced” tab and click to check the “Show Develop menu in menu bar” checkbox.



3. Close the “Preferences” window and return to Safari. There will now be a new item in the Safari menu labeled “Develop”. This provides many useful options for developers building and working with HTML pages. Click on the “Develop” menu and then click on the “Disable Local File Restrictions” option to check it.

This will allow Safari to display local HTML pages that contain Java Applets. This option is NOT needed to run Java Applets from sites over the web - it only applies to local files.



2.2 Runtime Options

There are 2 ways to run a system built with Corvid. Both will run the rules and logic the same way and it is generally easy to switch between the 2 modes at any time. Both options use Java to run the system, but one is run on the end user’s machine as a Java Applet and the other is run on a server as a Java Servlet and sends only HTML forms to the end user machine.

The way that the system is designed (variables, Logic Blocks, command block) is exactly the same regardless of how the system will be run or fielded. Applet or servlet make no difference on system logic design and development - they are only different in the end user interface and where the system runs. You can switch between runtime modes at any time as needed.

Applet

The applet approach is very easy to use since Corvid automatically builds a single HTML page that will load all the files needed to run the system. This HTML page and associated files can be simply put on a standard web server like any other HTML content and put online. However, this approach requires that the end user’s browser supports Java Applets. Some important browsers, specifically Safari on the iPhone and iPad, do NOT support java applets. Even PC and Mac browsers that do support applets, allow the end user to disable the feature on their machine, in which case applets will not work.

Java and the underlying Java support for all applets is created, managed and controlled by Oracle (previously by Sun Microcomputer). It is extremely well written, tested and validated with a huge user community. Despite this there have been security vulnerabilities found. Oracle has moved quickly to fix these, but the fact that they occurred has caused some users to disable Java Applets and for Apple to automatically display a warning message when a new applet is run. This appearance of a security problem, wether real or not, makes some end users uncomfortable running all Java Applets. This should be considered when deciding what will be best for the intended end user community of your system.

Servlet

The other way to run Corvid systems is via a Java Servlet. This still uses Java but in a very different way that does not have the potential problems of Applets. The system is run on a server under a "Servlet Container" program such as Apache Tomcat among others. Tomcat is a free download and can be easily installed on your Mac for Corvid system development. (Details are in chapter 14). The servlet sends only HTML forms to the end user's browser - not a Java Applet. All browsers, including the iPhone and iPad , support these forms and can run Corvid systems. Even if an end user has support for Java applets turned off, the servlet based approach will still work. Corvid makes it just as easy to build systems that use the servlet approach. In addition, when using the servlet approach the user interface and look-and-feel can be greatly enhanced by editing the HTML template files to add anything that can be coded in HTML.

While Tomcat and a servlet environment can be installed on your Mac at no cost and in only a few minutes, not all production servers or corporate web sites support Java Servlets - though it is becoming far more standard. If you are building a system that is aimed at distribution on a particular server or for incorporation in a particular web site, check that Java Servlets are supported. Corvid systems will work with Apache Tomcat, Glassfish, IBM Websphere or comparable programs.

Capability	Applet Runtime	Servlet Runtime
End User Interface Design		
Screen design language	Corvid Screen Commands	Corvid Screen Commands Templates can be edited with HTML / CSS / Java Script / Etc.
Degree of control	Control limited to color, position, fonts, images	Full control of entire screen - anything that can be done in HTML
Support for CSS, Java Script, Spry, Ajax, HTML5	None	Full
Support for Tables	Very limited	Full
Implement standard site look-and-feel	Set style properties that apply to all questions	HTML templates with replaceable parameters. Easy to fit into existing sites
System user interface	Applet window in HTML page	Full HTML page
Complexity of design	Limited - though more than enough for most systems	High - anything that can be done on an HTML page
Security		
Where system is run	End user machine	Server
System CVR (runtime) file sent to end user machine	Yes	No
Other system files	Must be available on server via a URL	Access to files can be blocked
Susceptibility to security vulnerabilities	Can and has happened, but monitored by Oracle	Much higher security

Capability	Applet Runtime	Servlet Runtime
Browser		
End user requirements	Must allow / support Java Applets	Any browser (Some browsers may not support advanced capabilities such as HTML5 or CSS3) Adobe Flash based systems require browsers that support Flash
Runs on iPad and iPhone	No	Yes
Commercial Systems		
Suitability	Somewhat limited	High - Recommended
Configuration		
Configuration to Use	Need to set Safari option on development machine	Need to install Apache Tomcat and install Corvid Servlet Runtime

So What to Use?

We at Exsys believe that the Java Servlet approach has many advantages over the Applet based approach, and any system intended for a business, organizational or professional use **should be fielded using servlets and the Corvid Servlet Runtime.** We have added many special features in Corvid to make development of Servlet based systems as easy as possible.

We only recommend the applet approach for fielding systems:

- Being built just to learn Corvid.
- In a classroom situation.
- For distribution in a controlled environment where java applets are known to be supported.
- When there is a requirement to field the system on a server that absolutely will not support servlets.

However, when first starting a new Corvid system, particularly when learning Corvid, the Applet based approach is easier to work with since it takes less setup and allows use of the much more extensive and powerful Trace option available with the Applet Runtime. The system can then later be converted to running with the Corvid Servlet Runtime with minimal effort.

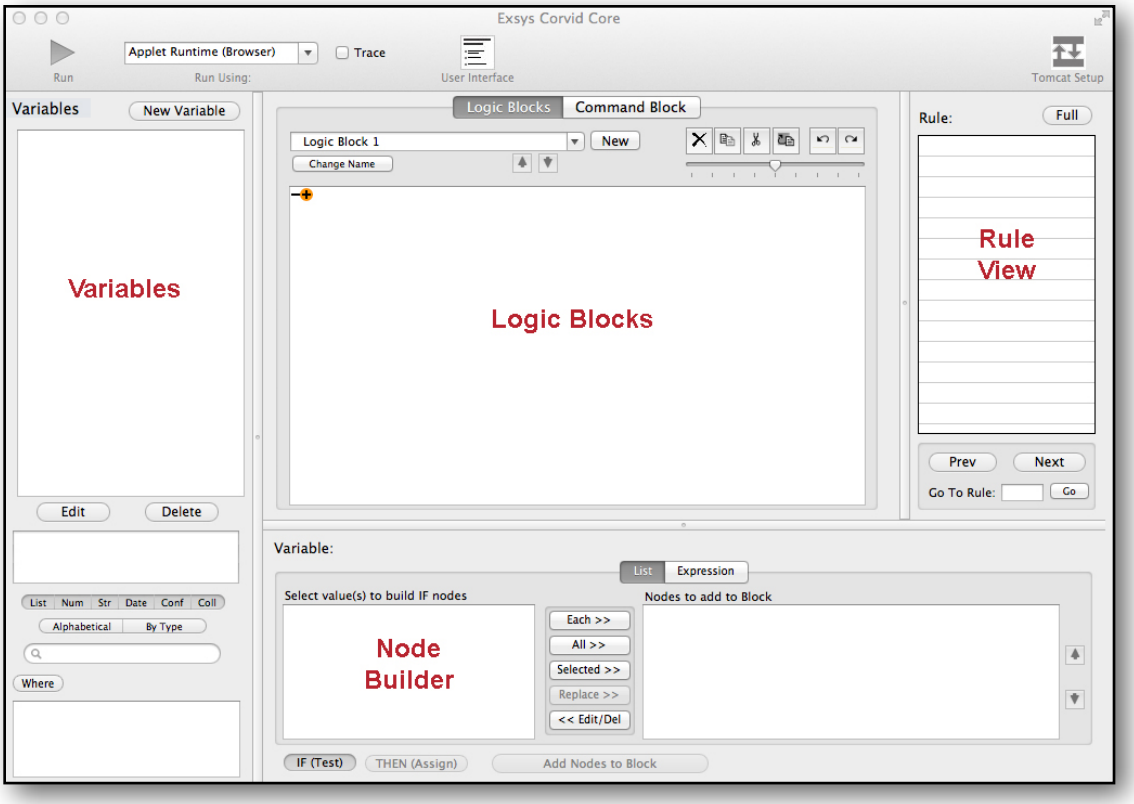
The User Interface look-and-feel options that apply to the running with the Applet Runtime will automatically carry over to the the servlet runtime and default templates. This can then be enhanced in the servlet environment by modifying the HTML templates to create far more complex interface.

3 Corvid's Main Window

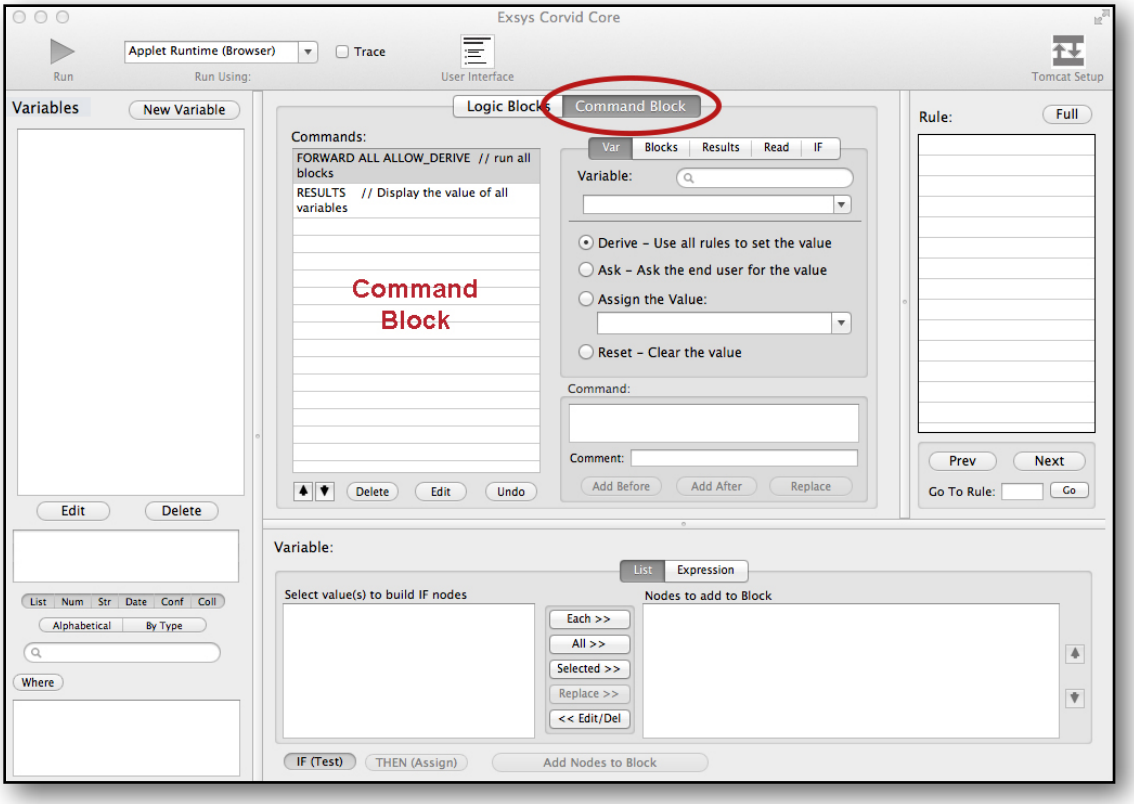
Corvid's developer interface has been designed as a single window Mac program. This puts all the items needed to build and edit systems on a single window that can be resized based on your screen. Other windows open for specific tasks, but most of the work is done in the panels of the main Corvid window.

The main Exsys Corvid Core window has 5 main panels:

- Variable panel for displaying, adding and editing variables.
- Logic Block panel for building Logic Blocks that define the rules in a system.
- The Node Builder panel for adding and editing specific nodes in the Logic Block.
- The Rule View panel for examining the rule produced by a branch in the Logic Block.



- The Command Block panel is for building commands that run the system.



The following chapters explain how to use these panels in building Corvid systems.

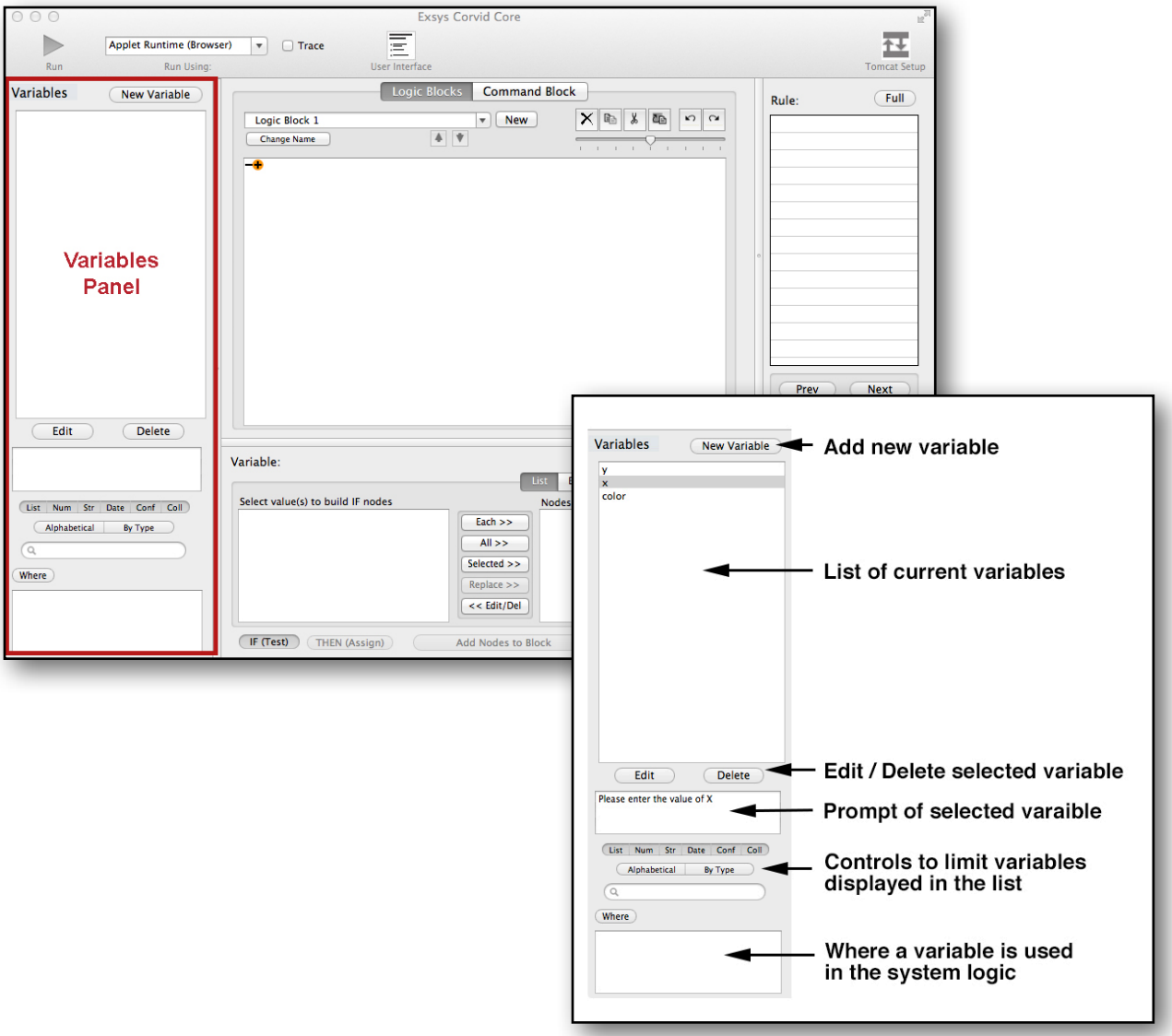
4 Variables

Variables are fundamental to all aspects of building Corvid systems. Corvid variables are generally similar to variables in standard programming languages and algebra. The logic of a system is defined with conditions and expressions - Boolean tests in the IF part and assignments in THEN part. Variables are used to create these IF and THEN conditions. All of the input asked of end users, all the intermediate calculations and results, reports and advice are all built using variables.

Variables have many options and special features whose usefulness will not be apparent until you start building systems. Many new Corvid users want to quickly get into the more interesting part of building rules and logic, but understanding variables is key to using Corvid effectively.

4.1 The Variables Panel

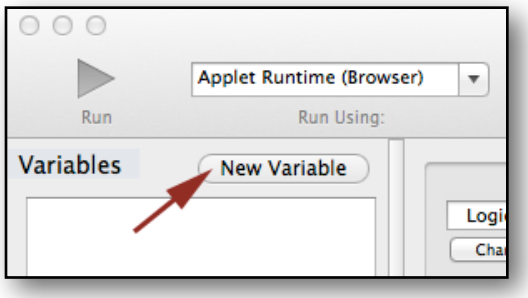
The variables in a system are displayed in the left panel on the main Corvid window. This displays the variables in the system and allows using, editing and adding new variables.



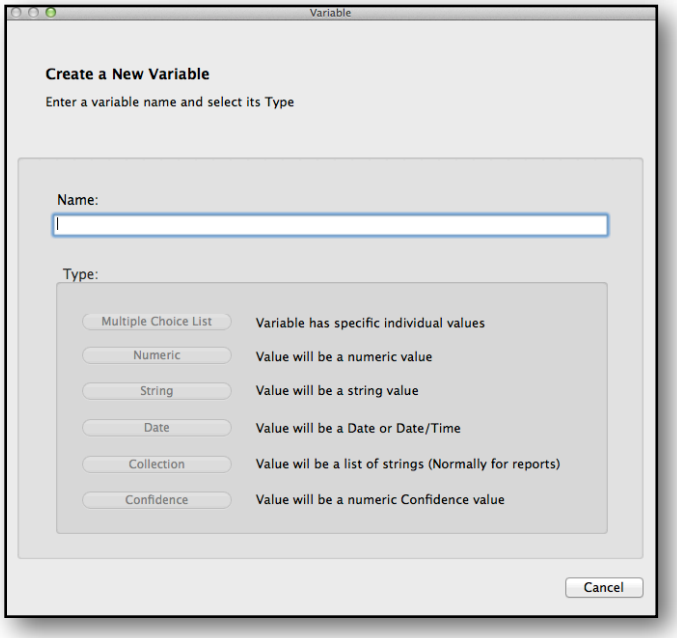
4.2 Adding Variables

New variables are added to a system by clicking the “New Variable” button

Variables can be added at any time. Some developers prefer to add many variables before starting to build the rules, others prefer to add variables only as they will need them in the rules. Either approach is fine. By default, Corvid will display the most recently added variables at the top of the variable list so they can be easily found and used.



When the “New Variable” button is clicked, the window for adding a new variable will be displayed. The a new variable must be given a unique name and specified to be a particular “Type” based on the type of data it will hold and how it will be used in the system.



4.3 Variable Names

All variables must have a name. There is no limit on the length of the name, but it must be unique and should be something that is easy to recognize and remember. Names are NOT case sensitive, so “WEIGHT” and “weight” are considered the same variable. Generally the name should be kept short, with a longer more detailed description put in the variable’s Prompt.

Names can only contain letters, numbers, a few special characters and most non-English characters. Spaces and tabs are not legal in names.

For systems built in English, variable names should only include:

A-Z a-z 0-9 _ # $ and %

Variable names built in non-English character sets can include any characters EXCEPT:

[~ ! @ ^ & * ( ) + = " ' > < . / : ; { } ? | \ `] - , <space> <tab>

Square brackets are used to indicate a variable in expressions, but should NOT be part of the variable name.

When naming a variable, any illegal characters (including spaces, tabs and brackets) will be automatically converted to the underscore character _.

NOTE: Corvid first converts any illegal characters to _ then checks that resulting name is unique ignoring case. So if there is a variable named TEMP_A, a new variable named Temp*A cannot be added since the * will be converted to _ and the names would match ignoring case.

The name of a variable can be changed later any time during system development by editing the variable. When a variable's name is changed, Corvid will automatically change it in all occurrences of the variable in rules, commands, etc.

The following are examples of typical variable names:

Color Todays_Date Price Country Length_of_beam Report

Within Corvid commands and expressions, variables are identified by the variable name in square brackets:

[Price] > 10

[Name] = "Exsys"

[Length_of_beam] < 58

4.4 Variable Types

All variables must also have a "Type" that determines what kind of data it can hold and what type of expressions it can be used in.

Corvid supports 6 variable Types:

Multiple Choice List	Fixed value list
Numeric	Numeric value
String	String value
Date	Date Value
Collection	Value is a list of strings (Used for reports)
Confidence	Value combines multiple Confidence values

Most of the Types are very similar to variables found in many programming languages, but some are specialized for building expert systems.

Multiple Choice List - Variables that can only be assigned one of a set list of possible values. This can be as simple as a value list of "Yes" and "No" or any fixed list such as states, day of the week, model types, etc. Multiple choice list variables should be used whenever the possible. Having a fixed set of possible values makes building and testing the logic much easier.

Numeric - Variables that will be assigned a numeric value. The value can be tested in algebraic expressions and assigned from complex numeric formulas.

String - Variables that will be assigned a string value. While string values can be tested and parsed with special functions, most string variables are used to get information that is just added to a report.

Date - Variables that are assigned a date and optional time. Like strings, these are generally to add content to a report. A simple function makes it easy to set the current date/time. There are date functions that can be used to test the time between dates which is sometimes needed in regulatory systems.

Collection - Special variables whose value is a list of strings. They can be though of like a shopping list where various rules may add something to the list, delete something from the list or test the values in the list. There are various ways these can be used, but primarily they are used to build reports. Collection variables are never directly asked of the end user, though they may have user input provided for other variable types added to them.

Confidence - Special variables used to build systems that handle probabilistic logic to find the "most likely" or "best" solution, even when there is not 100% certainty. The value of a Confidence variable is numeric, but it is the combined values of multiple assignments to the single variable.

The Type for a variable is determined by the type of data it will hold and how the variable will be used in the logic. The Type is selected by clicking on the associated button in the "New Variable" window.

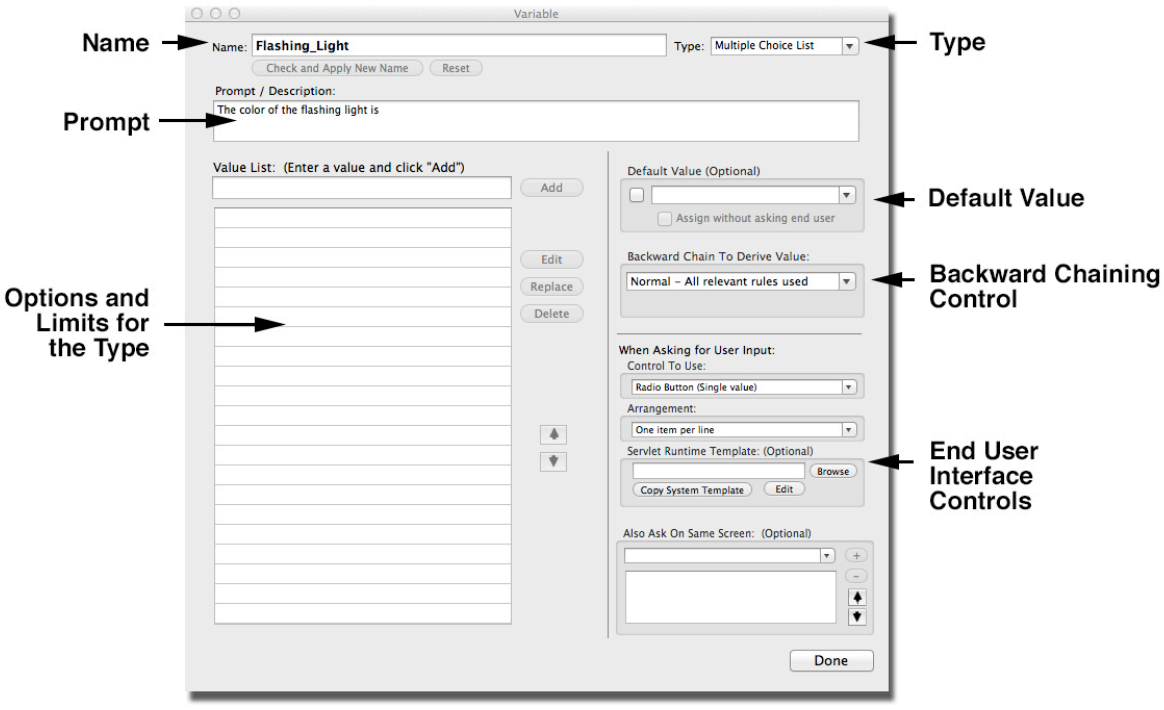
Choosing the correct Type will make system development much easier.

Once a type is selected, the window for setting the properties for that type of variable is displayed.

4.5 Setting Variable Properties

Depending on the Type selected for the variable, the window changes to allow setting the Prompt and the options for that specific type.

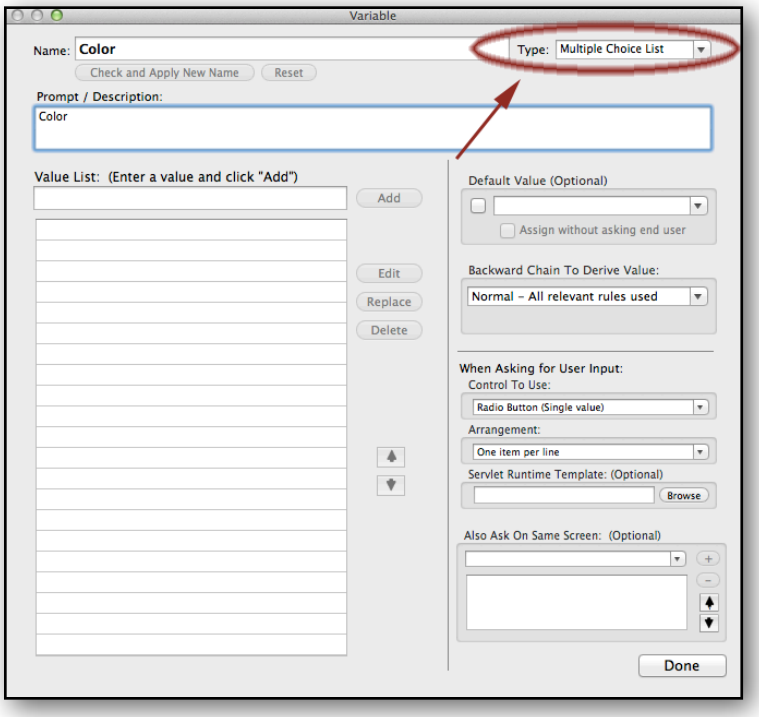
When Type button is clicked, the window for that Type will open. Each of the variable types have slightly different options, but all have a similar overall layout.



Unlike Name, the Type of a variable CANNOT be changed if the variable is used in a Logic Block to build rules.

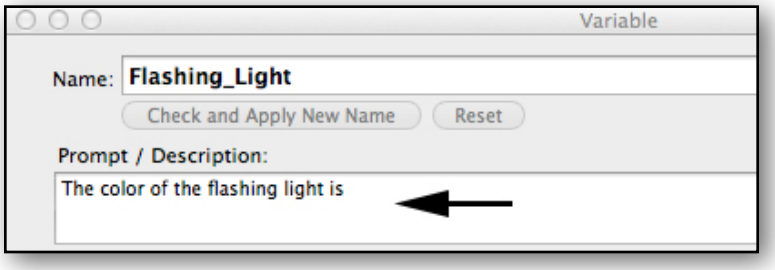
However, it can be changed immediately after a new variable has been added, or if it is not used in rules.

Click on the drop down list in the upper right and select the correct type. This will change the Type and display the window for setting options for that Type of variable. This can only be done before the variable is used in the system logic.



4.6 Prompt

In addition to the Name and Type, all variables have a “Prompt”. This typically explains what the variable represents. Normally the “Name” is relatively short and simple since it will be used in expressions, and the “Prompt” is a longer detailed description of what the variable means. The Prompt will be used when asking the end user for input and in reports. The Prompt can be any length and can include any characters. The Prompt can be edited directly in the Prompt edit box.



When adding new variables, the text for the Name will automatically be used for the Prompt. If the Name included illegal characters, these will be converted to _ in the Name, but will appear in the Prompt since it has no limits on allowed characters.

The prompt is not directly used in the system rules and is primarily for the end user interface to ask questions and display advice, and creating good prompts will make the system easier for end users to read and understand.

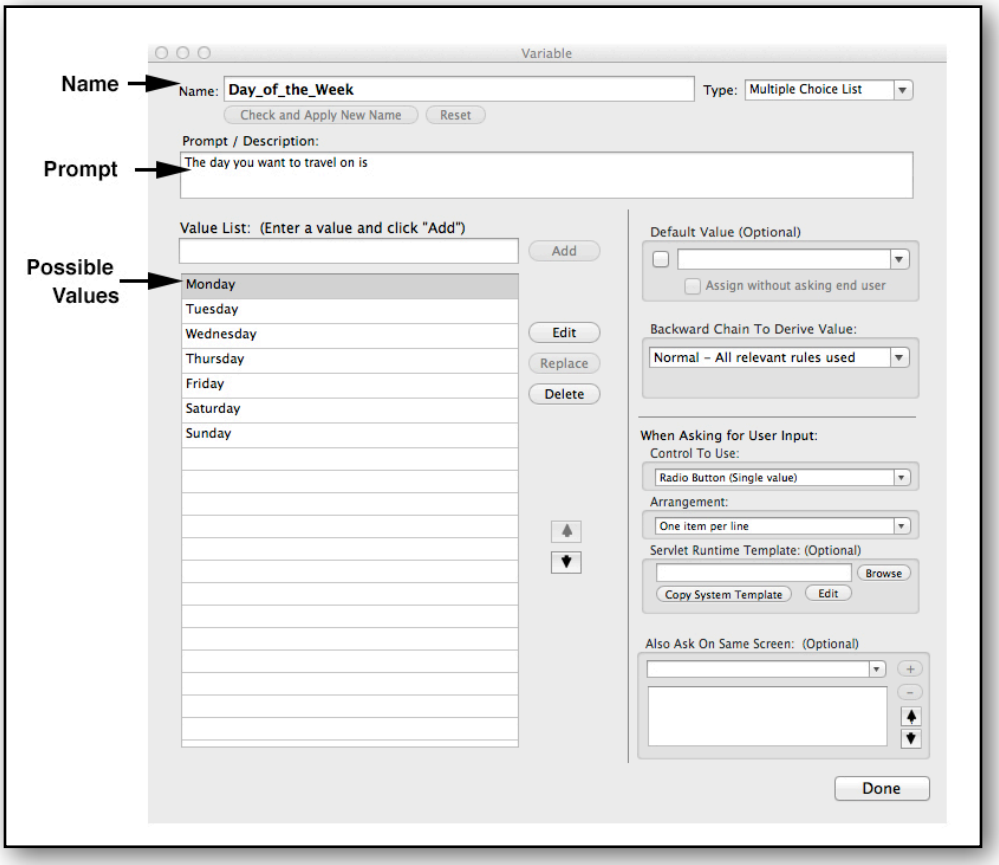
If a variable will be asked of the end user, the Prompt is the default text that will be displayed when the question is asked. It should clearly and precisely explain what is being asked, including any units or limits that might apply. For example, a variable might be named [Weight], but the prompt could be “The weight of the material being processed, not including the container, in kilograms. This should be a number between 0 and 100”. Depending on the desired end user interface, the prompt may be phrased as a question, such as “What is the color of the light?”.

The prompt can also be used in various other places in a system when something more than just the name of the variable is needed. Prompts are often used this way with Confidence variables that display system results and advice or when building reports to explain a value that has been assigned to the variable. In that case, the prompt will be more of a statement or explanation, rather than a question.

The Prompt can be easily changed any time during system development. When a new variable is added, the prompt will automatically be set to the name of the variable. It can be left this way during early development and then changed later when needed. If a prompt is only going to be used to ask for input. It may be best to phrase it as a question. (e.g. “What is the price?”). If it is going to be used in reports, it may be better to have it be a phrase which can be combined with a value to make a statement (e.g. “The total price would be”). There are also various ways to use text other than the Prompt when asking questions or displaying results.

4.7 Multiple Choice List Variables

Multiple Choice List variables (or just List variables) have a set list of possible values, and the actual value for a particular situation will be one, or more, of the values from the list.



List variables are one of the most commonly used of all Corvid variable types and should be used whenever it is possible to specify a fixed list of possible values for a variable.

Lists make user input easier since they just select an item from a list rather than entering text or a number. This greatly reduces the need to check user input for correct syntax, valid data, etc. Using a List variables also makes it easier to quickly build structured logic that covers all possible variable values.

Multiple choice list variables must have a list of values. Values can be any length and can include any characters, but generally should be kept reasonably short and easy to understand. The values should generally cover all possible values that the variable could have. There can be any number of values in the list.

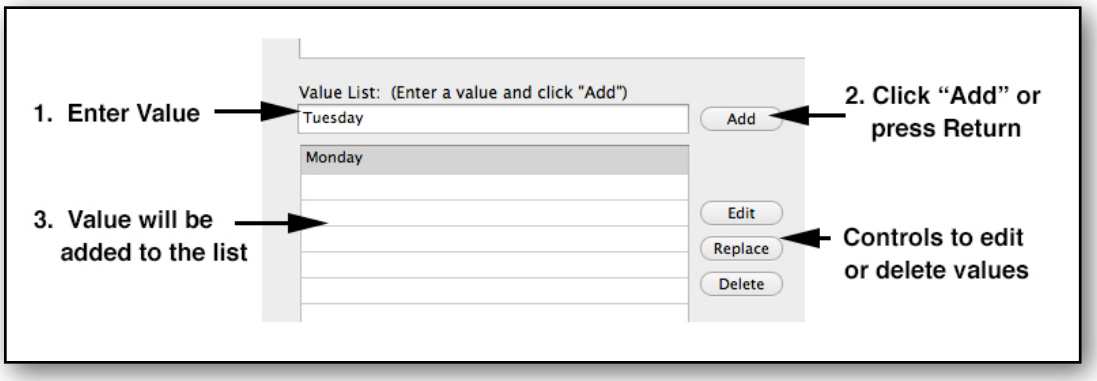
For example, the List variable [Day_of_the_week] could have a value list of:

Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

For List variables, a combination of the Prompt and one or more values should make an easily understandable statement. For example, the prompt for the variable [Day_of_the_week] could be “The day of the week you want to travel on is”. This combined with one of the values makes easily read statement.

Adding Values

Multiple choice list variables must have a list of values. Enter a value in the “Value List” edit box and press the return key or click the Add button. The value will be added to the list.



Repeat this process to add as many values as needed. Additional values can be added later, but it is best to have all the values in the list before building the rules.

Editing Values

The text of a value can be changed by:

- Click on the value to select it.
- Click the Edit button.
- Make any changes in the edit box.
- Click the Replace button.

A value can be deleted by selecting it and clicking the Delete button. If the value is in use in a Logic Block to build rules, it cannot be deleted until the Logic Block is edited to remove the use of that value.

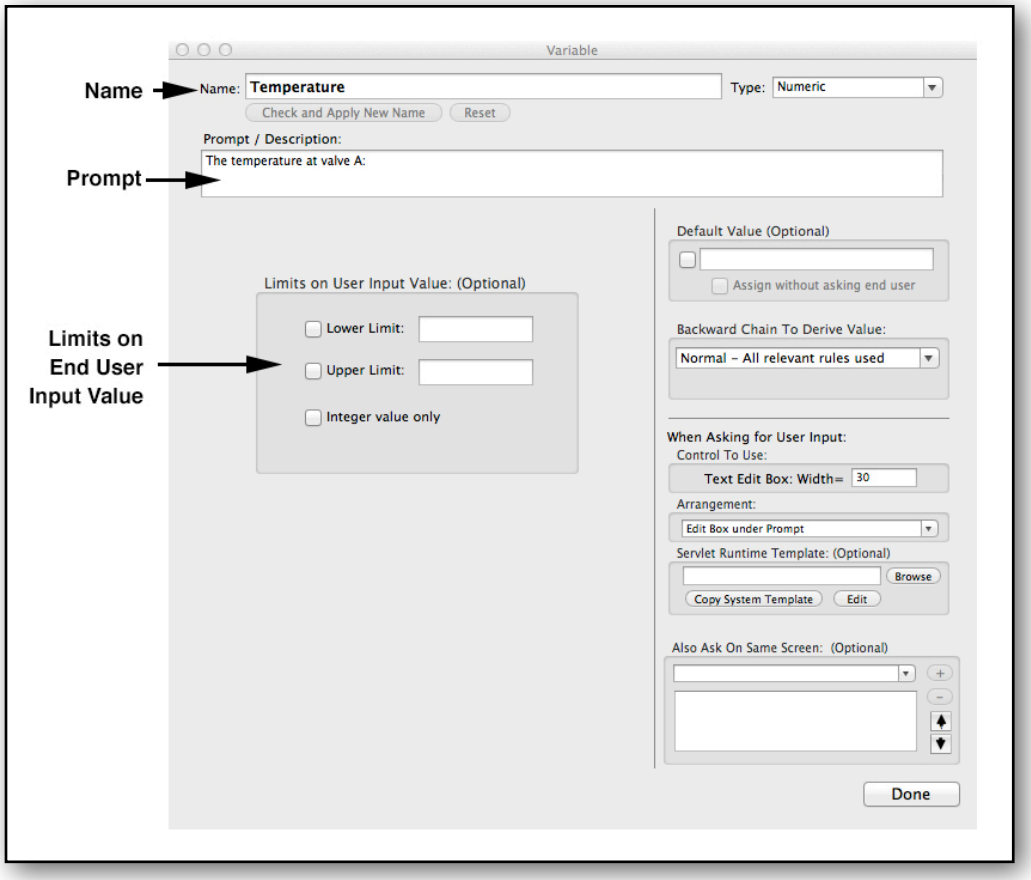
The order of the values in the value list does not matter in the logic, but it is the order that the values will be displayed in the end user interface when asking them to select a value. If you wish to change the order, select a value and click the up or down arrow button to move it.

4.8 Numeric Variables

Numeric variables are variables whose value is a number. The value is a “floating point” number, (e.g 123.456). Corvid numeric variables are very similar to numeric variables found in many computer languages. They are used in algebraic expressions and assignments anywhere a number can be used. For example:

[Price] < 10

[Area] = [Length] * [Width]



Like all Corvid variables, Numeric variables have a Name and a Prompt.

Options for the variable allow specifying specific upper and lower range limits that the value must be in, or that the value must be an integer. If the variable is asked of the end user, and they input a value that is out of the specified range, the value will be rejected and they will be re-asked to input a value.

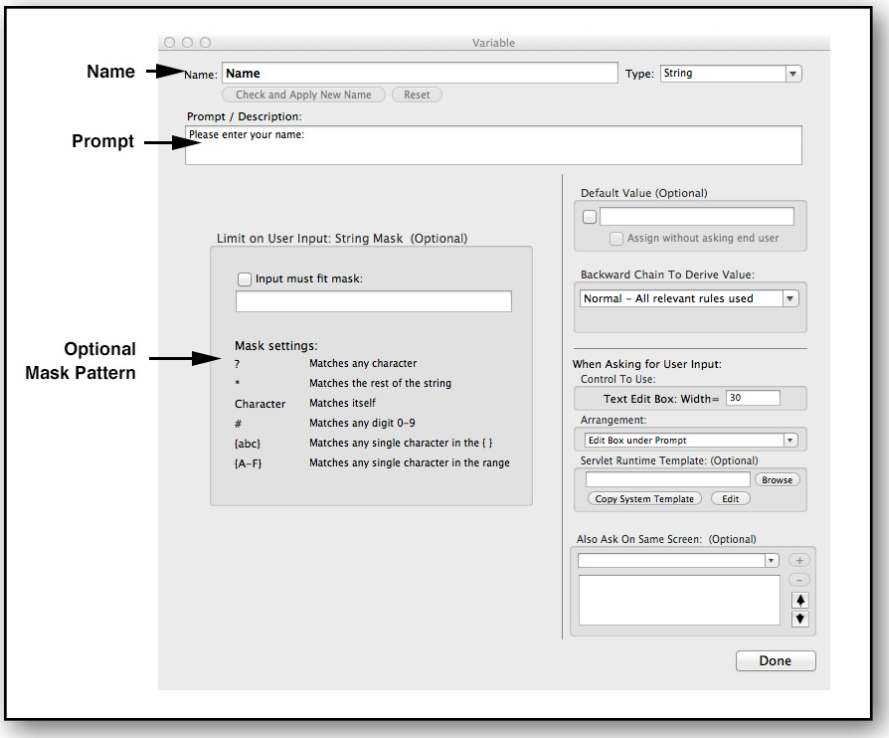
If limits are set, it is important to let the end user know what is expected, and the Prompt should include information on the limits and what values are allowed. Limits can also be used to make sure end users input is realistic so that the logic does not have to check for nonsensical input.

In some cases, a system may need a numeric value, but there are only a few specific possible values (e.g. Number of people in your car). In these cases, it can be better to use a List variable to cover the specific possible values.

4.9 String Variables

String variables are variables whose value is a text string. The value can be a string of any length. The String variable value can be used in string expressions to build, test or parse strings with various Corvid string parsing functions. For example:

[Name] = “John” + “ Doe”



String variables can have an optional “Mask Pattern” and end user input values must match the mask pattern or they will be immediately rejected and the variable re-asked.

Masks are specified by a string that indicates what character(s) are acceptable. This can be used if the input string has an expected pattern, and input not matching that pattern would be invalid.

Syntax	Matches
?	Matches any character
*	Matches the rest of the string, including spaces
Character	Matches itself
#	Matches any digit 0-9
{abc}	Matches any single character in the { }
{V-Z}	Matches any single character between V and Z

All character matching is NOT case sensitive. A mask of “a” will match either “a” or “A”.

For example:

To match a Social Security number, use **###-##-####**.

To match a product ID number that must start with an X or Y, followed by a number between 1 and 5 and then any other numbers, use **{XY}{1-5}***.

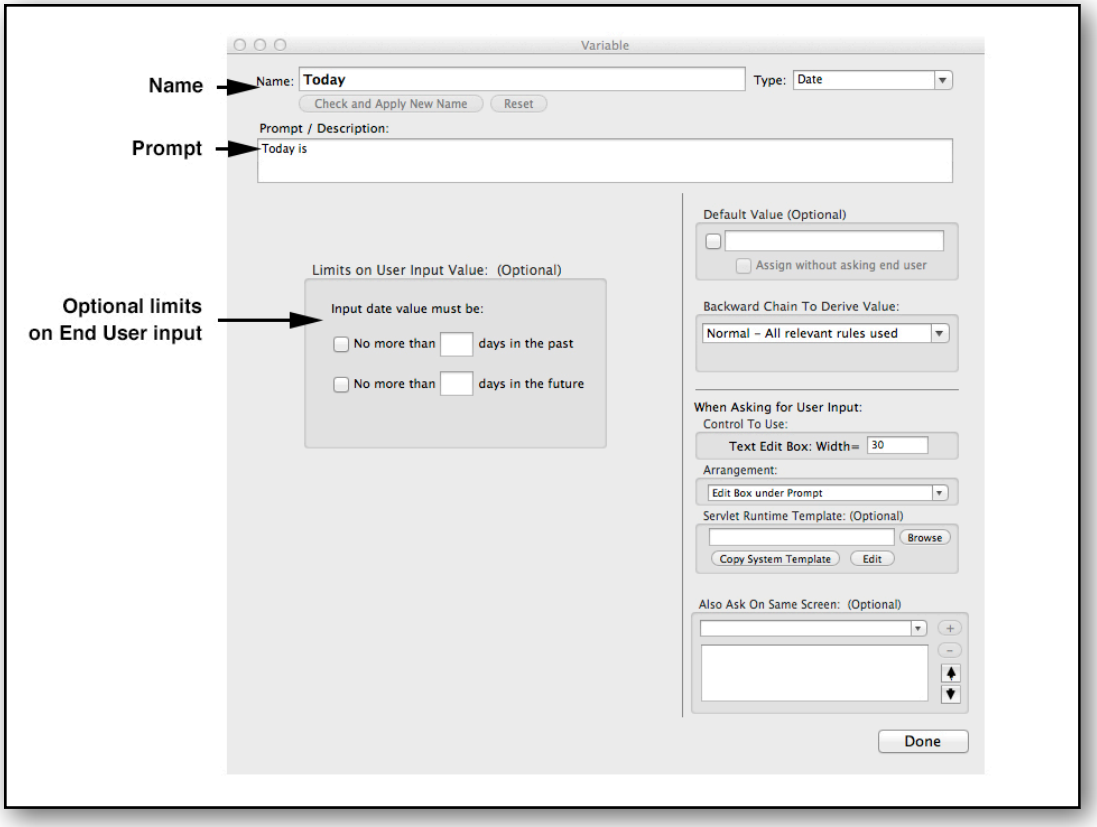
To match a 4 character string starting with Z, use **Z???**.

As with Numerics, if there are only a limited number of possible values for a string variable, it is better to use a List variable so that the value is better controlled and input is easier.

While string values can be parsed and tested in rule expressions, string variables are most often used for information that will not be analyzed by the system logic (e.g. name, address, phone number), and is primarily used for creating reports or other interfaces where free text is needed.

4.10 Date Variables

Date variables are variables whose value is a date and an optional time. Dates/times can be input by the end user, created by system logic or obtained from system clocks. Date values can be tested against other dates, or generated with Corvid’s date functions.



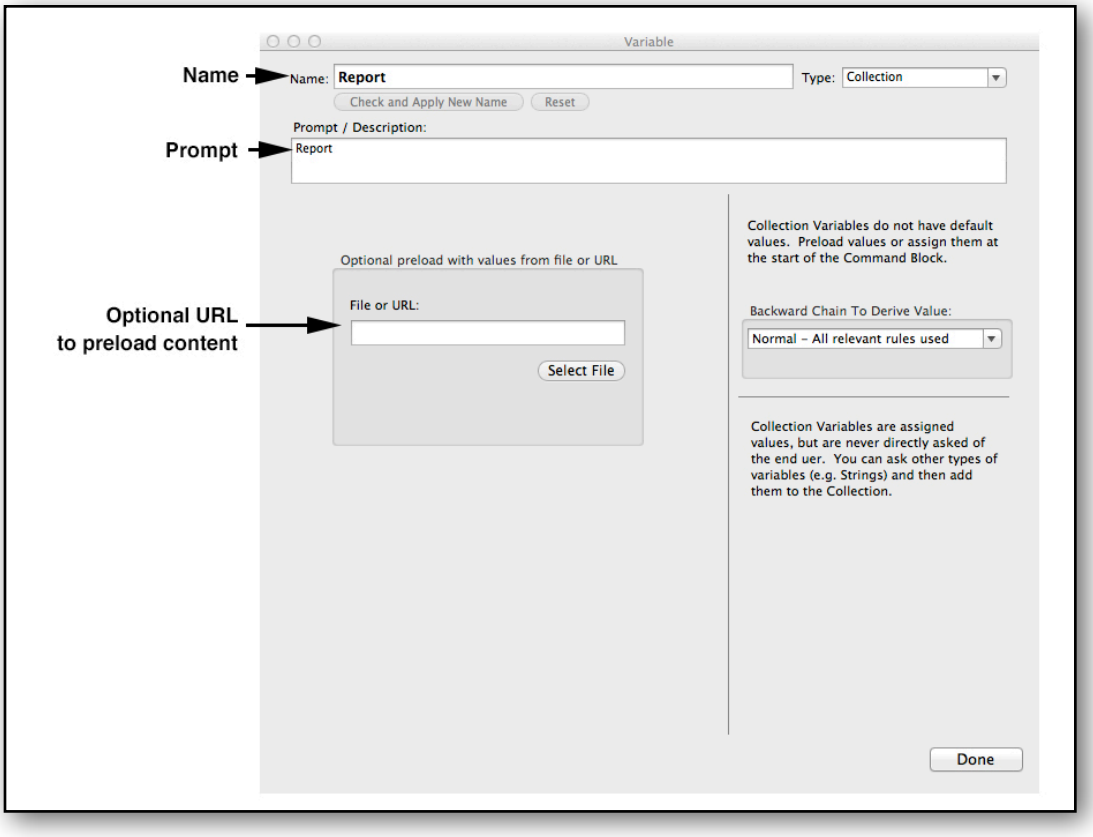
Options allow controlling the range of a date that an end user may enter. The date can be specified to be no more than X days in the future or past.

Dates can be complicated since there are many ways to specify dates (e.g. 8/21/12, Aug 21, 2012, August 21, 2012) and dates are not written the same way in all countries. In some countries the form is month/day/year and others it is day/month/year. This results in some string representations of a date being ambiguous and determined by the localization settings of the computer. This can be important for systems that may be run from multiple countries with different date representations.

The Corvid runtime programs use the Java date functions which are quite good at handling the many forms that a date can occur in. However, in cases where the date value is not going to be analyzed, and is only going to be added to a report as text, it can be better to use a String variable which can handle any text the user inputs.

4.11 Collection Variables

Collection variables are variables whose value is a list of text strings. They can be though of like a shopping list where various rules may add something to the list, delete something from the list or test the values in the list.



Collection variables are only assigned values - never directly asked of an end user. However the values assigned can come from user input or formulas.

Typically, each assignment adds to the list of strings. Corvid provides various options to add content to the Collection value list either in a sorted or unsorted manner. The overall value of a Collection variable can be many pages of text. The text can be used to build a report, and may include HTML or other types of content. The content in the Collection variable can also be test.

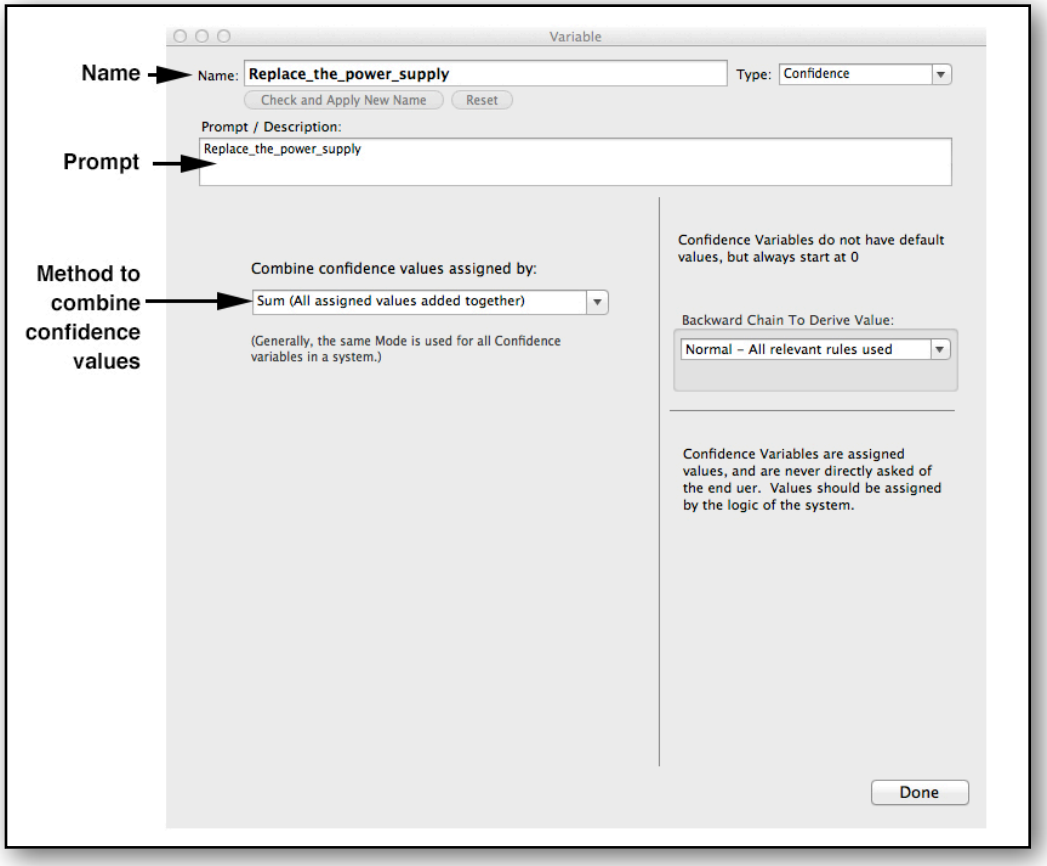
Collection variables are most often used to build complex reports, with various rules in the system adding pieces of content to the report, or the report can be built from a “template” file that combines report formatting with the values of other Corvid variables. This approach makes it easy to build reports that can be displayed at the end of a session.

A Collection variable can optionally be preloaded with the content of a file or from a URL. This is a very easy way to build reports using template files with embedded Corvid variables.

Using Collections variables to build reports is covered in chapter 15 on Reports.

4.12 Confidence Variables

Confidence variables are variables whose value is a numeric “Confidence value”. The value is a measure of how “good” or “likely” the diagnosis, advice or fact represented by the variable is.



Usually, this is used when the variable represents a possible recommendation or solution to the problem that the expert system solves, and the Prompt will be displayed as a recommendation if the variable’s Confidence value is highest or over some threshold.

Often, there will be a set of Confidence variables representing the various possible diagnoses or items of advice the system could give. The one(s) with the highest Confidence value are the ones most appropriate for, or consistent with, the user input for a specific session. Confidence variables allow building systems that weight and compare competing factors to select the most likely overall solution, and are used in probabilistic or “Fuzzy” systems.

Confidence variables can also be used in other ways, but in all cases, the variable will be assigned one or more numeric values which will be combined via a formula to produce the overall confidence variable value.

The value of a Confidence variable is a number and can be used in formulas any place that a numeric variable could be used.

Like Collection variables, Confidence variables are only assigned values - never directly asked of an end user. However the values assigned can come from user input or formulas.

During a session, a single Confidence variable will often be assigned multiple values by various rules. The values assigned may be specific values or calculated from formulas. These multiple values will be automatically combined to calculate an overall confidence value for the variable.

Confidence variables are somewhat unique to expert systems and behave differently than variables in other programming languages. In most computer languages if a variable has a value X and it is assigned a new value Y, the old value X will be lost and the new value will be Y. That is also the way Corvid string, numeric and date variables behave. However, Confidence variables remember and combine ALL the values they are assigned during a session to produce an overall value.

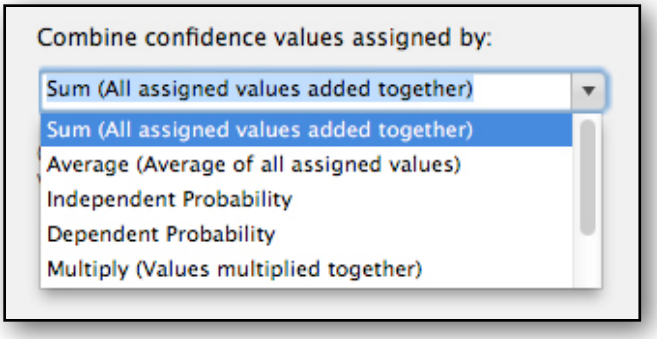
This allows multiple rules to independently contribute to the overall final confidence value. Some rules may indicate that the variable is a “good” solution and assign a higher value, while other rules may indicate it is a “less desirable” solution and assign a lower value. The combination of the various values assigned determine the overall value.

Trying to do this type of calculation using a normal numeric variable, would be quite difficult. Confidence variables are designed for it and do it automatically. They keep track of all the individual values assigned, along with an algorithm for how to combine them to produce an overall value.

For example, the simple default approach to confidence is the “Sum” method which just adds the individual values assigned together for the overall value. Individual rules can add or subtract “points” from the overall value by assigning positive or negative values to the collection. If 3 rules assigned values of 4, 7 and -6, the overall value would be 4 + 7 - 6 = 5. This is a simple approach, which works quite well for many situations.

Confidence Modes

When a new Confidence variable is added to a system, the mode (algorithm) used to combine values must be selected. Corvid provides 7 ways to combine the individual values. Each variable can have its own way of handling confidence allowing multiple techniques to be combined as needed. However, most systems using confidence will have a set of Confidence variables that all use the **same** mode to combine confidence values.



A mode should be selected that matches the approach used by the human expert providing the decision-making knowledge for the system.

The mode for combining values is selected from the drop down list. There are 7 options:.

> **Sum** - The values are added together. Positive values increase the overall confidence, negative values decrease the overall confidence. This is a simple system, but works very well for many systems. Unless there is valid statistical data, this is often the best way to combine “rule of thumb” factors in a decision.

Average - The values are added together as with “Sum”, and then divided by the number of values. This provides another simple way to combine competing factors, with individual factors having less influence when there are many values added.

Independent probability - The values are combined as if they were independent probabilities. If there are values X and Y, the combined value will be 1 - ((1-X) * (1-Y)) The individual values must be between 0 and 1. This is a statistically more rigorous approach, but requires that there be valid statistical data that can be applied.

Dependent probability - The values are combined as if they were dependent probabilities. If there are values X and Y, the combined value will be X * Y. As with the Independent mode, values must be between 0 and 1, and it requires having valid statistical data.

Multiply - The values are multiplied. This is the same as the dependent probability mode, but here there is no assumption that the values actually represent probabilities, and the values can be any positive value in any range. For example, a rule could lead to doubling the confidence by giving it a value of 2, or halving the value by giving it a value of .5. Values assigned in this system should be positive.

Maximum - returns the largest value assigned. This is useful for cases where individual rules can flag a variable as “good”, regardless of lower values from other rules.

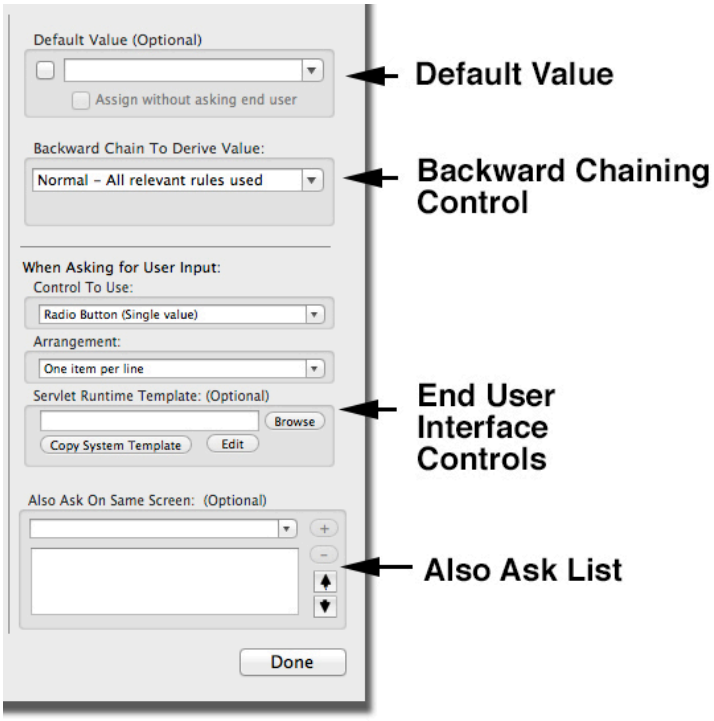
Minimum - returns the smallest value assigned. This is the opposite of Maximum. It is good when a rule can eliminate a variable by giving it a low value, regardless of high values given by other rules.

Selecting a Mode: The Sum system, while simple is adequate for most confidence systems and the “add or subtract points” approach is easy to work with and understand. If there is statistical data on a process, the Independent or dependent modes may work. If there are primarily thresholds for including items, the Maximum and Minimum approaches can work well. The goal is to use an approach that matches the way the human expert thinks about combining and weighting the various possible outcomes or items.

Confidence variables can also be used in formulas like any other numeric variable to assign a value to another Confidence variable. This allows the confidence values to propagate from one Confidence variable to other Confidence variables or to test the overall value assigned.

4.13 Right Side of the Variables Window

The right side of the variable window for each Type have similar options, though some Types do not have all the options.



Default Values

Variables that can be directly asked of the end user (List, Numeric, String and Date variables) can have Default Values. This is a value that will be preselected when the question is asked. The user can change the value, but to accept the default all they have to do is click the OK button.

In addition there is an "Assign without asking end user" checkbox. If this is checked, when the system would ask for the value, the default value will be immediately assigned **without asking the end user for input**. This can be very useful for variables that may have a value set from rules via backward chaining, but if no rules fire the default value should be used without asking the end user.

Backward Chaining Options

All variables have a backward chaining option drop down list. In most cases, this should be left at the default of "Normal - All relevant rules used". This is usually the best option and should be used when first starting a system.

Any backward chaining options that are set apply ONLY to the associated variable. Different variables can be set with different chaining options.

The backward chaining options are covered in chapter 7 on Backward Chaining.

User Interface Options

The variables that can be directly asked of the end user (List, Numeric, String and Date variables) have options for the type of control to use when asking for input and the arrangement of controls relative to the prompt. These allow customizing how a variable will be asked.

List variables have the most options for the type of control to use. Numeric, String and Date variables must use an edit box, but the width of the edit box can be controlled based on the length of the input string expected. All can have the control either put on the same line as the prompt, or on the line under it.

These options are covered in more detail in chapter 12 on User Interface.

The control and arrangement options always apply to running with the Corvid Applet Runtime, and to running with the Corvid Servlet Runtime when the default question templates are used. However, the options can changed and extended when using the Corvid Servlet Runtime by editing the HTML code used to ask the question.

Also Ask

List, Numeric, String and Date variables can have an optional Also Ask list to ask multiple questions at the same time on the same screen.

This can be done by setting the properties for the “Controlling” variable. This is the variable that is being asked first, and which triggers the other variables to also be asked.

When this controlling variable is asked, the other variables will be asked on the same screen, unless they have already been asked for some other reason - either asked individually, or a part of an Also Ask with another variable. Variables that already have been asked will NOT be re-asked even if they are in an “Also Ask” list.

To add other variables that will be asked on the same screen:

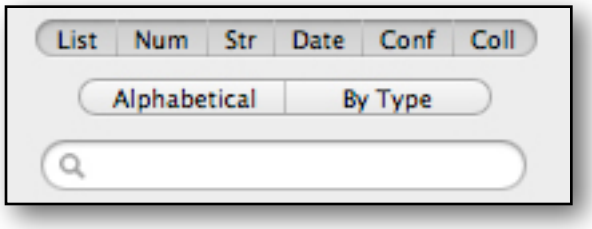
- In the Also Ask drop down list, select a variable to ask on the same screen.
- Click Add.
- Continue adding more variables as needed.
- Variables in the Also Ask can be reordered by selecting them and clicking the Up and Down arrow buttons.

The variables in the Also Ask list will be asked with whatever controls and options are set for those individual variables.

4.14 Working with Variables

All the variables in a system will be listed in the variables panel. By default they will be arranged in the order they were added with the most recently added variable will be at the top. This makes is easy to find a new variable when it is added to the system.

The variables can also be displayed sorted alphabetically by clicking the “Alphabetical” button, or displayed sorted by type by clicking the “By Type” button.



To limit the list to only certain types of variables, select or deselect the Type buttons. If a Type is not selected, variables of that type will not be displayed in the list.

To search for variables with specific text in their name, prompt or value list, enter the search text in the search box and click the search icon. The list will be limited to the variables that meet the search test. **Remember, if the list is limited with a search string, the search string MUST be erased to again see all the variables in the list.**

The Where button at the bottom of the Variable panel will display where in a system a variable is used. Select a variable in the list and click “Where”

4.15 Selecting the Type of Variable to Use

Often the meaning of a variable, and the data it will hold, makes it easy to select a variable’s type. However, occasionally multiple types would work. The following are some pointers to use when having difficulty selecting a type:

- If the possible values can be listed, use a List variable. This should always be the first choice if possible.

- If the value is to be asked of the end user and is only going to be directly incorporated into reports or results (not analyzed or processed by the system logic) use a String variable. Strings provide the maximum flexibility for user input.
- If the value needs to be analyzed using algebraic expressions or assignments that require a numeric value, use a Numeric variable. Setting limits on the range of the input value can simplify the logic. Remember that end users can be quite "creative" in their input and the values should be restricted or checked in the logic to make sure they make sense.
- If the value needs to be a date or time to use in expressions or functions that require a date value, use a Date variable. When date variables are used, be sure to take into account the various date formats and localizations possible. If the date is not analyzed and only handled as text, use a String variable instead.
- If the value will be tested or parsed with the string parsing functions, use a String variable. This is also the appropriate option if a string value must be built up from various pieces, however if the content of the text string is long or has multiple items, a Collection variable may be a better option.
- If building a formatted report or a report that will be built up by various sections of content, use a Collection variable. Collections are very flexible and powerful variables ideally suited to reports.
- If a system will select the "most likely" solution/recommendation from among a group of possibilities, use a group of Confidence variables. These will allow building sections of logic that contribute to the overall probability of each Confidence variable. The one(s) with the highest Confidence value will be the "best" recommendations.

4.16 Starting a System

When starting a new Corvid system, you must add at least one variable to be able to start to build the Logic Block. Often it is best to start by adding multiple variables at the start.

If the system is being built from existing documentation or logic diagrams, start by creating the system variables by copying a pasting text from the documentation. This makes it much faster to build the Logic Blocks since all the variables are defined and ready to use. Additional variables can be added later at any time as needed.

Generally the variables that are asked of the end user have an obvious Type based on the data they will hold. There are various approaches to creating systems and the output variables usually are Confidence or Collection variables, though any Type can be used when needed.

5 Rules and Expressions

The decision making steps in a Corvid system are defined by IF/THEN rules. All programming languages have some form of IF/THEN statement, but Corvid rules are conceptually quite different. They are heuristic rules that each describe a small step or aspect of an overall decision making task.

Unlike IF/THEN statements in computer code that conditionally restrict when other code will execute, Corvid rules are not “hard coded” together, where specific lines of code are only executed when the IF condition is true. Instead they are individual “rules of thumb” that are implicitly linked together by what information is needed and relevant to reaching a conclusion. Rules may be relevant, and used to reach a conclusion or provide advice, but only if needed. Corvid rules are more like the explanation that an expert would provide if asked how they reached a particular conclusion.

Because of this, the variables in a Corvid system are not used the way they would be in a computer program. They are instead used to build the descriptions of independent factors (rules) that can be used to make a decision, and the Exsys Inference Engine will then use the rules as appropriate to run the system.

Rules are built using Logic Blocks which provide a way to organize and structure related rules. The next chapter covers how to build rules in Logic Blocks, but here we will look at the individual expressions which make up the boolean tests and assignments used in rules.

Corvid rules are made up of one or more boolean IF conditions that each evaluate to True or False, and one or more THEN conditions which assign a value to a Corvid variable. When the IF conditions in a rule are true, the assignments in the THEN part are made. When there are multiple IF conditions in a rule, they must **ALL** be true for the overall IF part to be true. (The IF conditions are ANDed).

```
IF:
        Boolean expression 1
and:    Boolean expression 2
and:    Boolean expression 3
        ....
THEN:
        Assignment of value to variable 1
and:    Assignment of value to variable 2
and:    Assignment of value to variable 3
        ...
```

5.1 IF Conditions

The IF conditions (nodes) in the Logic Block trees are always boolean tests. There a 2 main types of tests that can be built - those using List variables and everything else.

List Variable IF Conditions

IF nodes using List variables are built by selecting a List variable and one or more of the variable’s values.

The condition will be:

```
variable_name = value
```

or when multiple values are selected:

```
variable_name = value1, value2, value3, ...
```

Unlike expressions, in this case the variable name is NOT in square brackets. This makes List conditions easier to read, and distinct from expressions.

When the system is run, if the specified value has been selected due to user input or other system rules, the condition will be true. If there are multiple values in an IF condition, the values are combined with OR and the condition will be true if **any** of the values are set. For example:

color = blue, green

will be true is the selected value is either blue or green.

Other Variable IF Conditions

IF condition built with other (not List) variables are very similar to boolean expressions in most programming languages. Generally the IF conditions have 3 parts:

Expression Operator Expression

The expressions are Algebraic and can be made up of Corvid variables, constants, functions, etc. The operators are usually =, >, <, >=, <=, !=.

In expressions, Corvid variables are indicated by the variable name **in square brackets [ ]**. Parenthesis can be used when needed for more complex expressions. For example:

[x] > 0 will be true if the Corvid variable named "x" has a value greater than 0, otherwise it will be false

([temp] + 20) / 5 = [x]*3 will be true if the left and right expressions evaluate to the same value, otherwise false.

5.2 THEN Conditions

THEN conditions are always assignments of a value to a variable. When the system is run, if the rule is used, and the IF conditions are all true, the assignments in the THEN nodes will be made.

As with IF conditions, List variables have a special, more readable syntax and all other variables use a more algebraic approach.

List Variable THEN Conditions

THEN nodes using List variables are built by selecting a list variable and one or more of the variable's values. These look like List variable IF conditions, but are assignments rather than tests. The color coding in Logic Blocks makes these easy to differentiate.

The THEN condition will be:

variable_name = value

or when multiple values are selected:

variable_name = value1, value2, value3, ...

For List variables, the variable name is NOT in square brackets. If there are multiple values a List variable THEN condition, they are **ALL** made and the variable will have all the values assigned.

Other Variable THEN Conditions

THEN condition assignments for all other variables are are:

[varname] = expression

where the expression evaluates to a value that matches the type of the variable. A numeric variable must be assigned a numeric value, a string variable must be assigned a string value, etc.

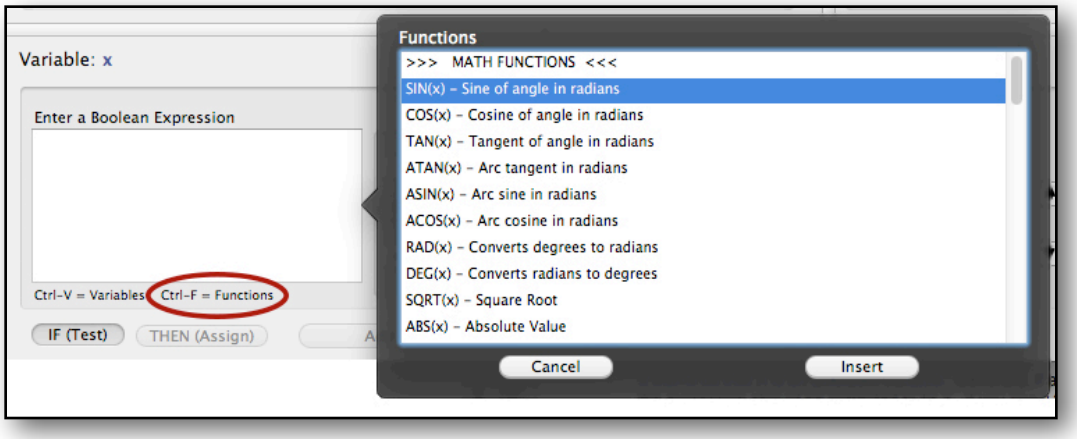
5.3 Collection Variables

Collection variables have a somewhat different syntax for IF and THEN nodes since instead of a single value, a Collection's value is a list of strings. Collections use methods and properties that allow testing the list in various way and adding values to the list in different locations. These are covered in Appendix B.5 on Collection variable properties.

5.4 Functions and Operators

Functions

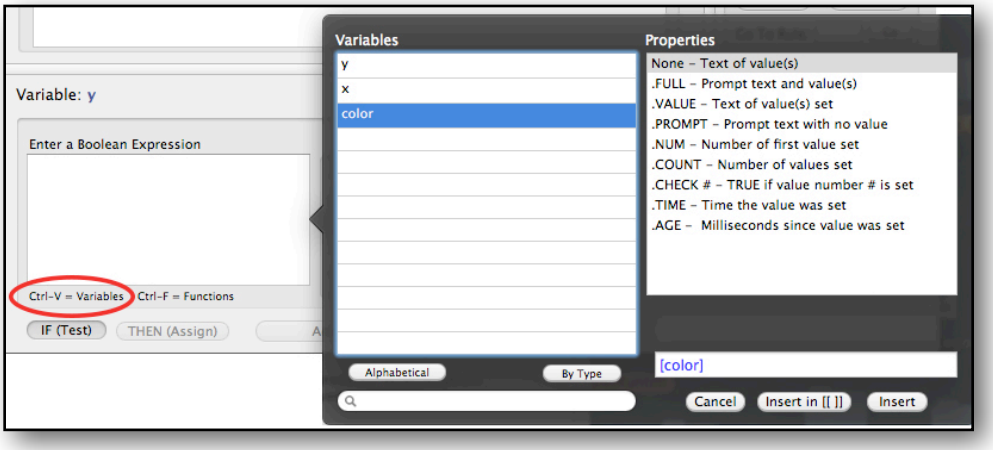
Corvid supports a wide range of functions for building expressions. When building an expression in an edit box, hold the **Control key down and press the F key** - to display the list of functions,. This will display a popup window listing the functions. This includes the standard trig and log functions, but also includes special functions for string parsing and date manipulation. Each function has a short description of its meaning. A detailed explanation of the functions can be found in Appendix A of this manual.



To add a function, just double click on it in the popup. The function will be added to the expression with placeholders for the arguments that the function takes. Just edit the arguments to whatever is needed in your situation. Any edit box that supports the Function popup will be marked "Ctrl-F=Functions" under the edit box to remind you.

Variables

Corvid also will display a list of the variables when building an expression in an edit box. Hold the **Control key down and press the V key**. This will display a popup window listing the functions.



Just double click on the variable and it will be added to the expression. When a system has many variables, or the variables have long names, this is very convenient. The other options for variables and Properties are cover in Appendix B.

Logical Operators

Within the Boolean expression the logical operators:

!	NOT
&	AND
!	OR

can be used to build more complex logical expressions that evaluate to True or False.

For example: (([X] < [Y]) | ([X] > [Z])) & ([Y] >= [Z])

Arithmetic Operators

All of the standard arithmetic operators can be used.

Operator	Meaning
+	Addition For stings, concatenation
-	Subtraction
*****	Multiplication
/	Division
^	Exponentiation (3^2 is 3 squared)
%	Modulo - Remainder after integer division (7 % 3 = 1)

Order of Precedence

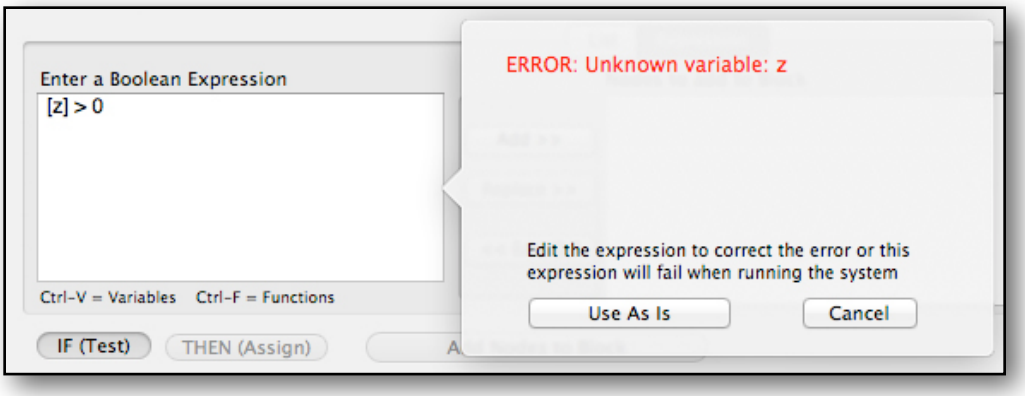
The precedence order in evaluating expressions is the usual one: raising to a power has the highest priority and is followed by division and multiplication, then addition and subtraction. Expressions are evaluated left to right. However, use of parenthesis is strongly encouraged to make expressions clear, easy to read and to avoid confusion.

For example: The order of precedence rules will evaluate **a/b*c** as **(a/b)*c**. If you wish to instead have it evaluated as **a/(b * c)**, use parenthesis to change the order that the operations are performed. Generally it is good to include parenthesis even if it does not change the standard order of precedence. Other developers reading or maintaining a system may not be familiar with the order of precedence rules and could misinterpret the way complex expressions will be evaluated. Parenthesis can be nested to any degree needed.

Syntax Checking

When an IF or THEN condition is built and added to the system. Corvid does a syntax check to make sure the expression is valid. It will check for a wide range of errors in mismatching type, illegal variables, etc.

It cannot check for some types of errors that only occur at runtime, but it will catch most common errors.



Here It warns us that Z is an unknown variable. Generally, if a syntax warning is displayed, the expression should be corrected to fix it. Just click it in the edit box, make any changes and add it again. When you click in the edit box, the warning popup will disappear. If you know the expression is actually OK (for example, if the variable is reported as unknown, but will be added momentarily), click the “Use as Is” button to add the expression despite the warning. However, unless the necessary changes are made, there will be a runtime error.

6 Logic Blocks

Corvid systems are made up of IF/THEN rules that describe the various steps in making a decision. Logic Blocks are the way you create, define and organize those rules.

Logic Blocks let you organize and group related rules into structured tree diagrams that make it easier to build complete and maintainable systems. Each row in a Logic Block makes a Rule.

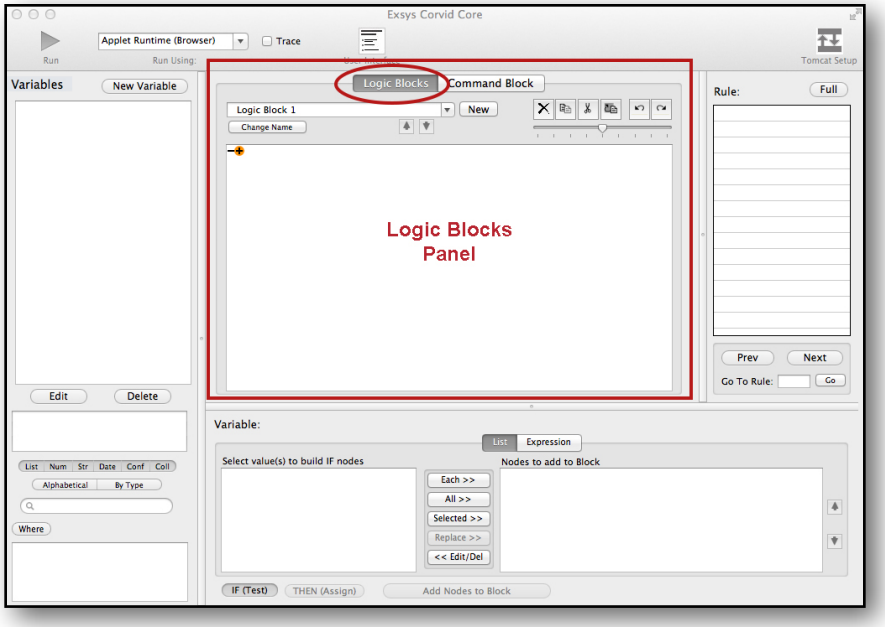
Important: What the Inference Engine uses is the rules - it generally does not really matter how you structure the Logic Blocks as long as all the needed rules are created somewhere. The Logic Blocks should be structured in the way that make the most sense to you.

There are many equivalent and correct ways to build the Logic Blocks for a system. Generally the best approach when there is existing documentation of the decision making process, such as decision trees, regulations or other diagrams, is to try to organize the Logic Blocks to match the existing documentation. That will be easiest and allow matching of the Logic Block to the existing documentation to find any gaps.

When there is no existing documentation to follow, the best approach is to have a Logic Block describe an “aspect” of the decision-making task - though how large or small that aspect is, is your choice. When there is a group of rules that all relate to setting a variable used in other rules, it can be convenient to put them in their own Logic Block or blocks since they are related, but even this is not required. Always try to follow the approach and methodology used by the human expert providing the knowledge that is being captured in the system.

Building Logic Blocks in pretty much up to the way you want to approach a problem and very different from traditional programming. This is because the Logic Block is NOT traditional computer code, it is generating rules that will be processed by the Exsys Inference Engine - which will combine the rules to drive the analysis in the system and the end user interaction.

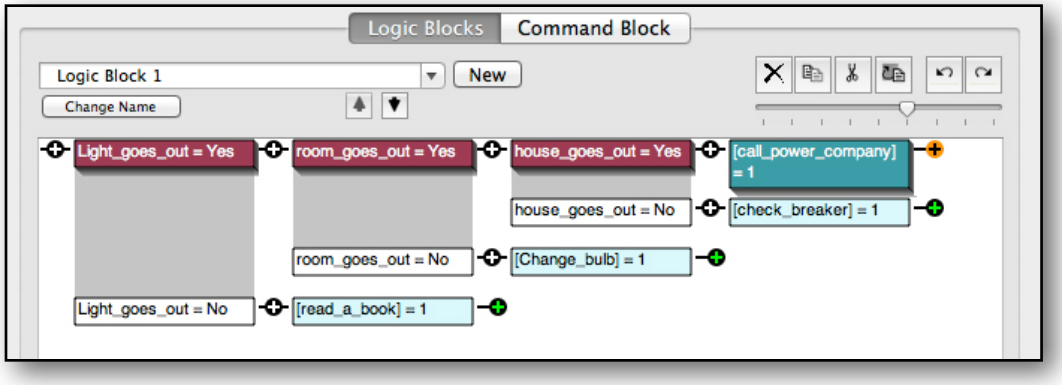
Logic Blocks are built in the Corvid Logic Blocks panel. This is in the center of the screen. Make sure the Lock Blocks tab is selected.



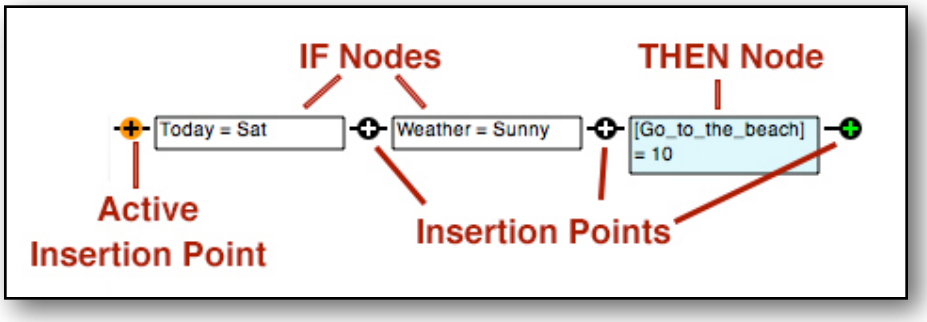
A system can have as many Logic Blocks as it needs.

6.1 Logic Block Structure

Logic Blocks are fundamentally tree diagrams. Each horizontal row defines a rule.

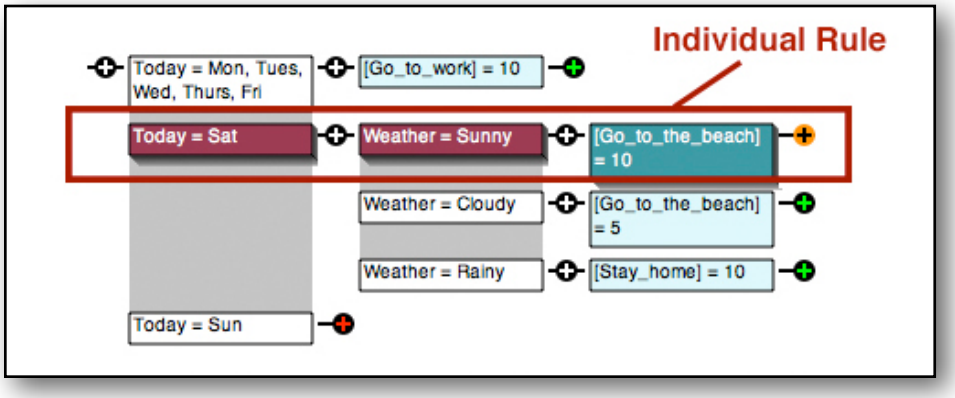


There are IF nodes that are boolean test conditions and THEN nodes that are assignments of a value to a variable. IF nodes are displayed in white boxes with black letters, or maroon with with letters when selected. THEN nodes have a light blue background with black letter, or teal blue with white letters when selected.

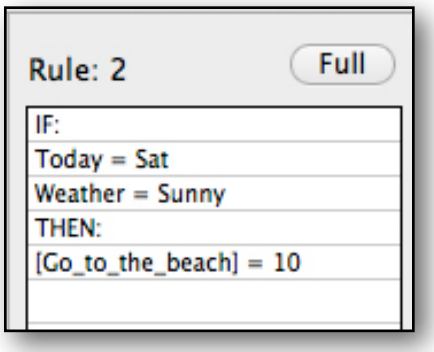


Between the nodes are “Insertion Points”, indicated by a dot with a + on it. Insertion points are used to add nodes to the tree and to examine the nodes that lead up to that point. There will be one insertion point highlighted with an orange dot that indicates the “Active Insertion Point” where new nodes will be added.

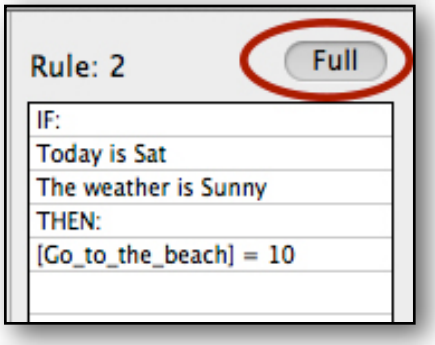
Each row in the Logic Block defines a rule.



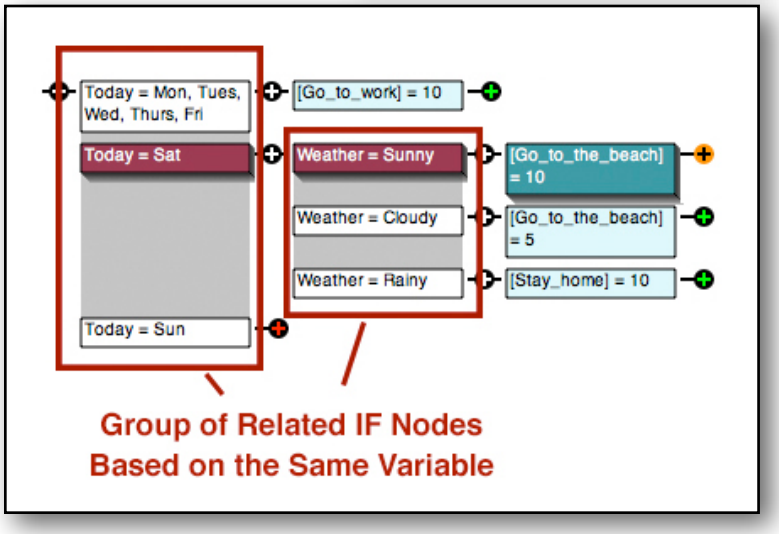
The rule will be all the IF and THEN nodes from that line. Clicking on an insertion point highlights the nodes that will makeup the rule leading to that insertion point. When nodes are highlighted, IF nodes are displayed as a maroon background and THEN Nodes have a darker blue background. In addition, the Rule View panel to the right will display the associated rule in an IF/THEN form.



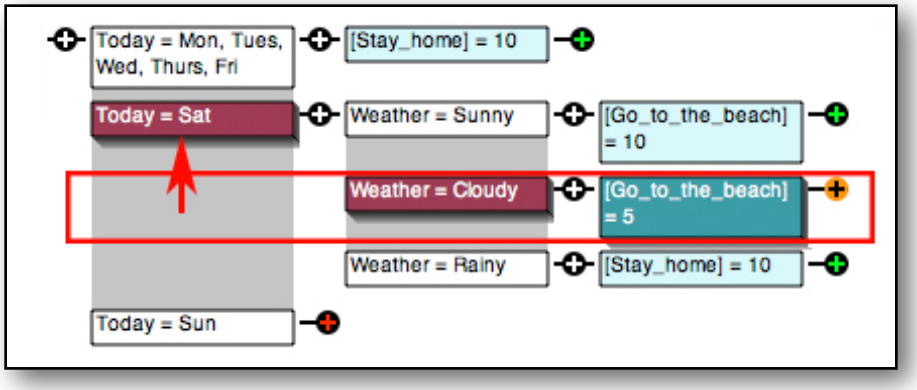
The Rule View panel can display the rule using the compressed version of the IF and THEN conditions used in the Logic Block, or can be made easier to read by clicking the “Full” toggle button to display the full text for List Variables. This will convert the nodes to a combination of the Prompt text and values in a more readable form. The Full option has no effect on formulas and expressions.



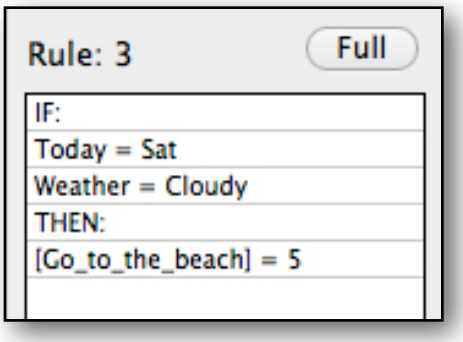
The real advantage of Logic Blocks over just building individual rules is that the tree structure makes it easy to organize rules so that they cover all possible logical cases. If a List variable has 3 values, it is very easy to simultaneously add 3 branches to the tree to cover these. If a set of formulas are needed to cover the logical options these can be added at the same time to create multiple branches. When multiple IF nodes are added at the same time, they are grouped with a gray background to indicate that they are based on the same List variable or were added as a group of expressions.



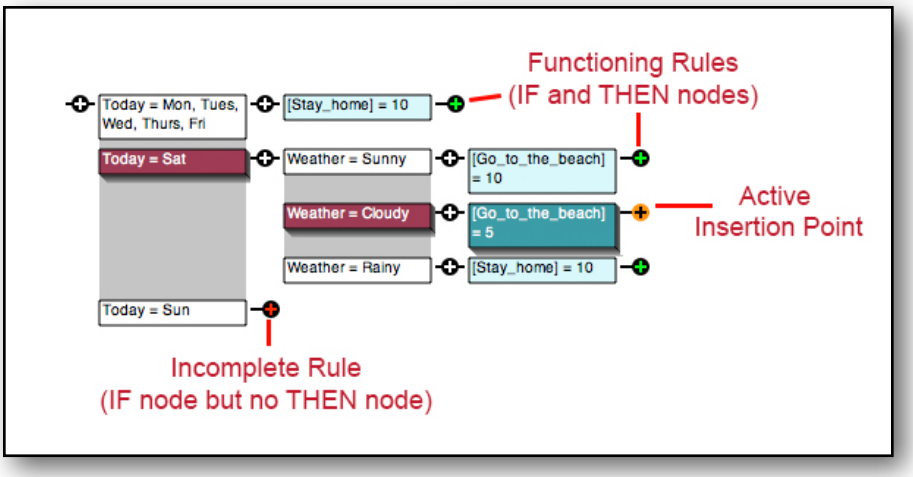
As the tree expands with new nodes, the gray areas between related nodes also expands. Each row in the tree is still a rule, however when a gray area between nodes is reached, the relevant nodes is the one **above** the gray area.



Corvid makes this easy to see by highlighting the IF and THEN nodes that make up the rule to the selected active insertion point, and displaying the rule in IF/THEN form in the Rule View panel.

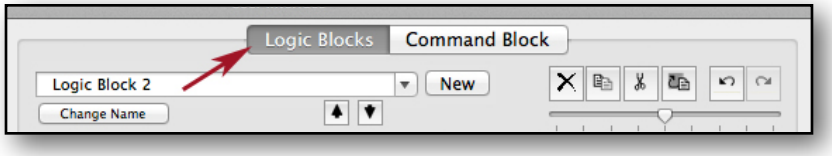


The last insertion point on a row will have either a red or green + on it. If it is red, that means that the row cannot build a rule because it has IF, but no THEN nodes. When at least one THEN node is added the + will become green. (Note, the insertion point becomes green when the **first** THEN node is added, your logic may require multiple THEN nodes on a row, and just because the insertion point is green does not mean you have necessarily added all the nodes your system will need. Additional THEN nodes can always be added.)

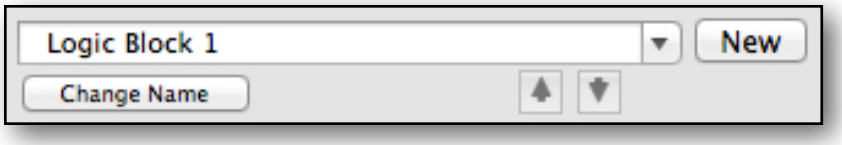


6.2 Logic Block Controls

The Logic Block panel shares the same space on the screen as the Command Block panel. These are selected by tabs at the top of the panel. To work on Logic Blocks, make sure the Logic Block tab is selected. (Command Blocks are covered in Chapter 11)

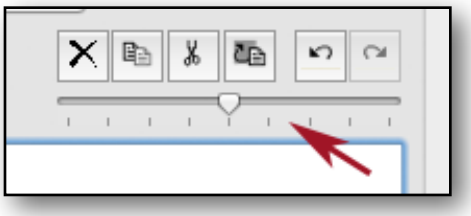


When Corvid starts, it has a single empty Logic Block, named “Logic Block 1”. If you have opened an existing system, it will display the first Logic Block and if you want to see another, just click on the name drop down and select the one you want. When there a multiple Logic Blocks, you can also go the next or previous one by clicking the up and down arrow under the Logic Block name.

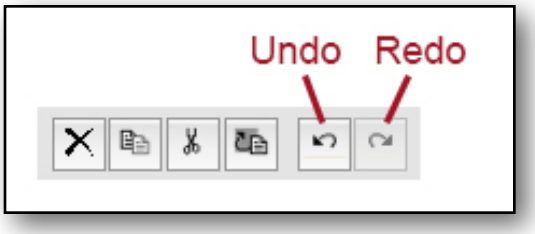


New Logic Blocks can be added by clicking on the New button. This will create a new Logic Block named “Logic Block n” where n is the number of the new Logic Block. The default names can be used, but when a system has more than a few Logic Blocks it is recommended to change their names to ones more indicative of what the Logic Block represents. The Logic Block name can be changed by clicking the “Change Name” button under the name drop-down. Enter the new name and Corvid will automatically apply the change to all rules and commands that reference that Logic Block.

The size of each node and the number of nodes displayed can be controlled by the scale slider above the upper right corner of the Logic Block. Moving this to the right makes the nodes larger and displays fewer nodes in the panel. Moving to the left makes the nodes smaller and displays more of the Logic Block. When the nodes are quite small, the text is not displayed, but clicking on a node or insertion point will still show the text in the Rule View panel.



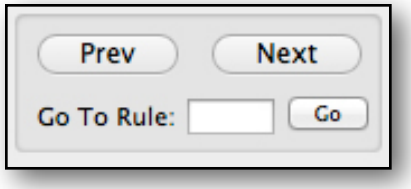
When a change is made to the Logic Block, the Undo button allows stepping back to the previous state of the block before the change. If an error is made, this is a quick way to move back. Either click the Undo button, select Undo from the Edit menu or click Command-Z.



Undo can be multiple times to step back to earlier states of the Logic Block. If Undo is clicked too many times, the Redo button will return the state prior to the last Undo. Redo can be done clicking the Redo button, selecting Redo under the Edit menu or clicking Shift-Command-Z.

The buttons left of the Undo button are for Cut, Copy and Paste in the block and are covered in section 6.9 on editing Logic Blocks.

Under the Rule View panel on the right are “Prev” and “Next” buttons. These provide an easy way to step through the rules in the Logic Block. They select the previous or next rule in the Logic Block. This highlights the nodes for that rule and displays the rule in IF/THEN form in the Rule View window. These are a very convenient way to examine the rules in a Logic Block.



The “Go To Rule” edit box below the Prev/Next buttons allows you to jump to a specific rule. The rule number is just the row number in the Logic Block. The “Where” command for variables and other options refer to specific rule numbers and this is one way to jump to those.

6.3 Nodes in a Logic Block

When Corvid starts, it has a single empty Logic Block, named “Logic Block 1”. Additional Logic Blocks can be added by clicking on the “New” button next to the Logic Block name.

The first node in a Logic Block must be an “IF (Test)” and the THEN button will be disabled.

The rule being built should describe a step or a factor to consider in the decision making process. This may come from human expert or existing documentation. This type of rule that describes a single step or factor is called a “heuristic”. Generally they can be though of a simple “rules of thumb” that the expert intuitively combines to make a decision. However, if you ask they expert how or why they came to the conclusion they did, they would explain the individual factors as heuristic rules.

The rule will be made up of one or more IF boolean conditions that evaluate to True or False and one or more THEN conditions that assign a value to a variable. The IF conditions are ANDed together, so **all** must be true for the rule to be true. The rule will be built using the nodes on a single row of the Logic Block. The boolean condition in the left-most node will the first (top) IF condition in the rule. The next node will be the next IF condition, etc. Likewise the THEN nodes will create THEN assignments in the rule from left to right.

Most rules that have multiple conditions will have some conditions that are more general than others, and the more detailed conditions should only be tested if the general ones are true. This is easy to implement in Corvid. Just put the more general conditions on the left and add more detailed conditions dependent on the general one to the right.

For example, a rule might be:

IF:

The problem is related to the printer

AND: The issue with the printer is it is not printing

AND: The power light on the printer is not lit

AND: ...

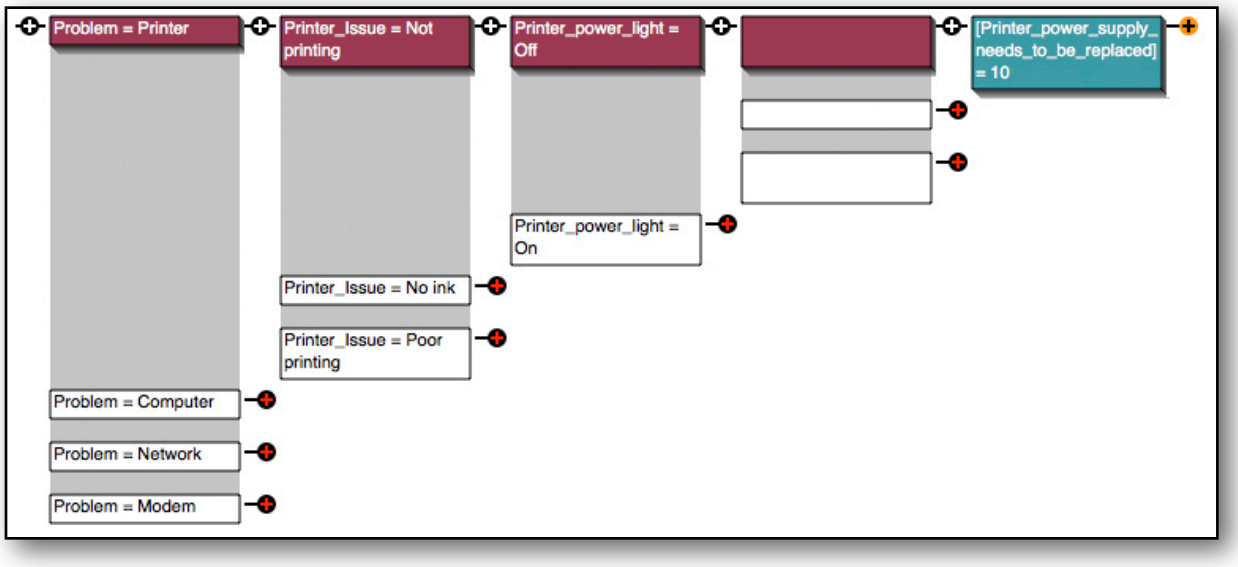
THEN:

The printer power supply needs to be replaced

The system should only test the 2nd and 3rd conditions when the problem is related to the printer, and should only consider the power light when the printer is not printing.

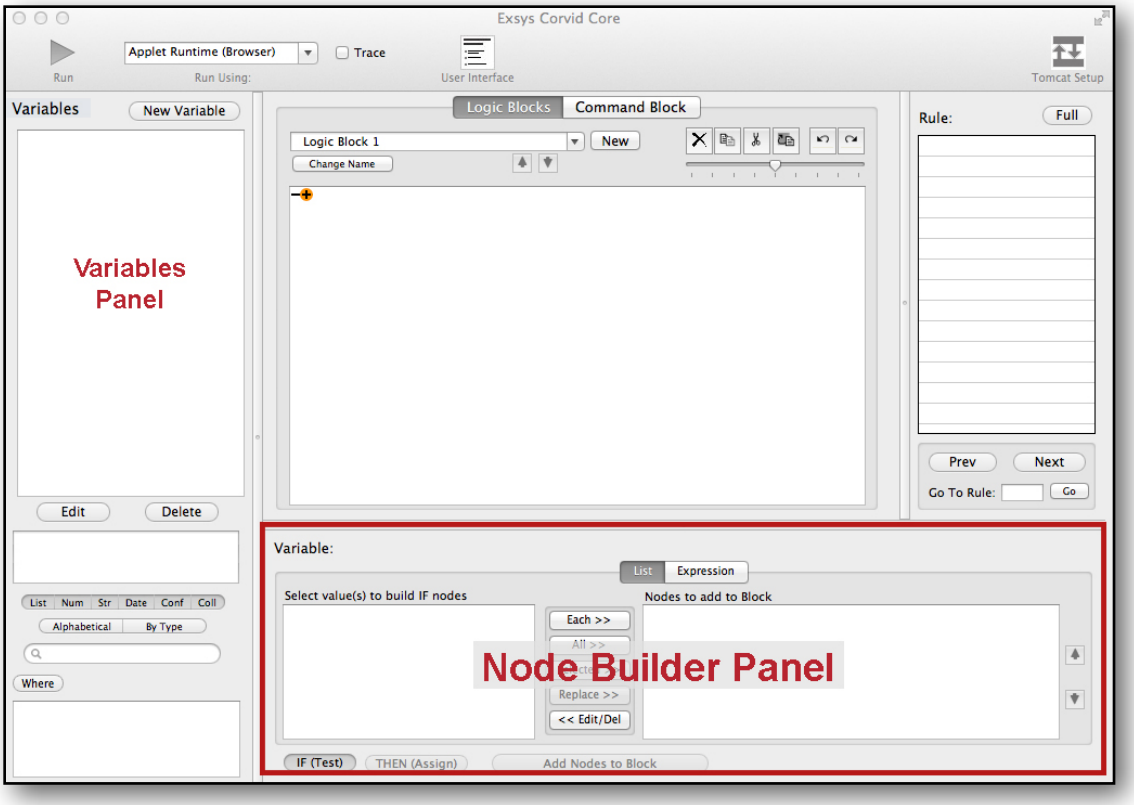
The Corvid Inference Engine will automatically test the rule this way. It will derive or ask the end user for the information needed to test the rule from the top down. If a condition turns out to be false, the overall rule is false and the rest of the conditions will be ignored, and it will move on to another rule. (However, when the Inference Engine already has enough information to know that a lower level condition is already false, it will skip the rule without testing any other higher level conditions)

Building the Logic Block should be done by first adding the top condition(s) as nodes and then building to the right for the more detailed conditions. If during development, a node needs to be inserted between nodes, that is easy to do by just selecting the insertion point between the nodes and adding it there.



6.4 Adding Nodes to the Logic Block

Nodes are added to the Logic Block in the Node Builder panel at the bottom of the window. This is used in conjunction with the Variables panel to select the variable to use to build the node.

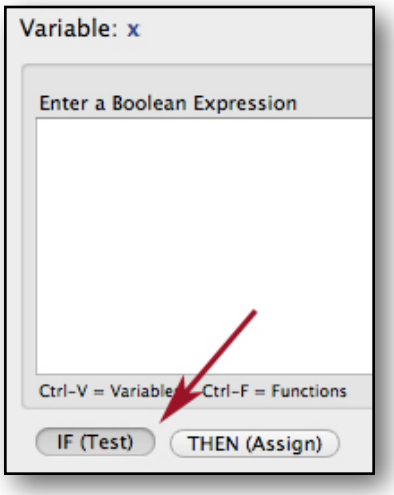


Most nodes should be added in a group that covers various possible cases or user input values. This will expand the Logic Block tree by adding (or expanding) a branch for each of the nodes added. Adding branches for each logical case makes it easy to see the various options and fill in a tree that covers all cases.

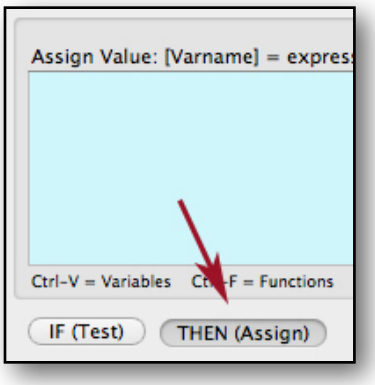
IF or THEN

The fist step in adding a node is to make sure the IF / THEN buttons are set correctly for the type of node to be added. These are that bottom left corner of the Node Builder Panel. Just click the button for the type of node to add.

Corvid will automatically disable one of the buttons when the active insertion point requires only a specific type of node. For example, a new Logic Block must start with an IF node and the THEN button will be disabled. Corvid will also prevent inserting an IF node between 2 THEN nodes or a THEN node between 2 IF nodes.



When the THEN button is selected, the edit box becomes light blue (same as the THEN node background) to remind you that a THEN node is being added.



Select the Variable to Use

The next step is to select the variable to use to build the node in the Variables panel. For List variables, the Node Builder panel will display the variable's value list.

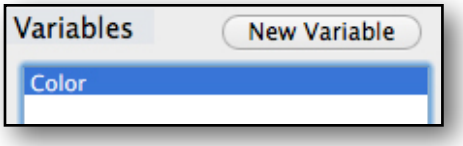
For other types of variables, the variable will be copied to the expression builder edit box.

List Variable Nodes

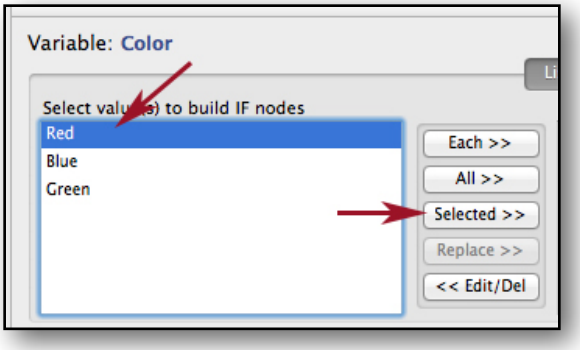
List variables are the easiest type to use to build IF nodes. This is because they have a fixed set of possible values and all that is needed is to select which value(s) belong on each branch of the tree.

To add an IF Condition with a List variable:

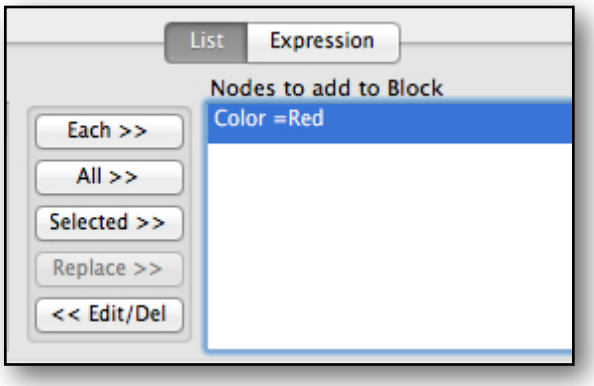
- Click on the List variable to use in the Variable list.



- It's associated value list will appear in the node builder list.
- Select a value or values. (Use shift click to select a block of values and Command click to select multiple individual values).

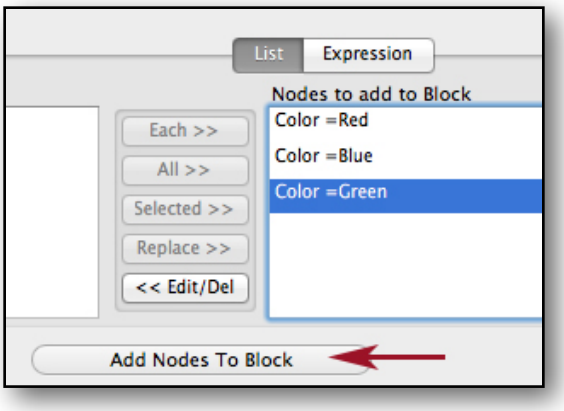


- Click the Selected button to add that condition to the "Nodes to add to Block" list on the right.

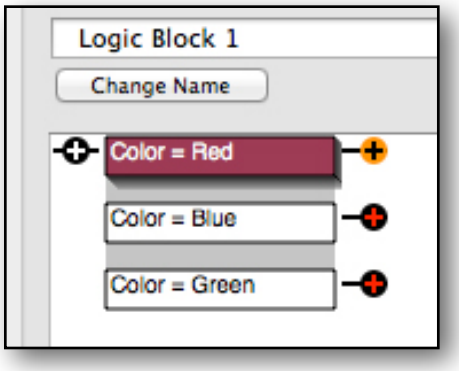


Repeat the steps above to add each value or group of values to the Nodes to Add list.

If multiple values have the same logical meaning or consequences, they should be combined to build a single condition rather than multiple identical rules. When multiple values are added in a single node they are combined with **OR**, meaning that **if any of the values is true, the full condition will be true**.



- Once all the conditions have been added, click the “Add Nodes To Block” button to add the group of nodes into the Logic Block at the Active Insertion Point. These will be added in the same order as they were put in the “Nodes to Add” list, and the active insertion point will move to the right of the top node added.



- It is not required to build nodes with each of the values. If a value has no logical meaning or consequences, it can be skipped. If the end user selects that value, it will not trigger any rule.

Since multiple IF nodes are generally added at the same time, Corvid has 2 special button to simplify the process.

- The “Each” button will build one IF condition for each of the values in the value list
- The “All” button will build a single condition combining all values in the value list with OR.

The “Selected” button adds the value or values selected in the value list. The “Each” and “All” buttons can be used with the “Selected” button build groups of conditions. To quickly add a branch for each value, just click the “Each” button and then the “Add Nodes to Block” button.

Conditions in the “Nodes to Add” list can be changed by selecting a condition in the “Nodes to Add” list and clicking the “Edit / Del” button to move those values back to the left value list.

The “Replace” button is used when editing a node and allows a new value(s) be used to replace the selected value in the “Nodes to Add” list.

The nodes will be added to the Logic Block in the same order that they appear in the list. In most cases, Corvid and the Inference Engine do not care about the order, but if you are trying to build a Logic Block that matches an existing decision tree or just prefer to structure the Logic Block a particular way, the order of the nodes can be changed by selecting a node and clicking the up and down arrows to the right of the “Nodes to Add” list.

6.5 IF Conditions Using Algebraic Expressions

The other type of IF conditions that can be added to Logic Blocks are boolean expressions built as algebraic expressions with any type of variable. Here instead of just selecting a List variable's value, it is necessary to build Algebraic expressions that will evaluate to True or False.

There will be a left and right part of the expression separated by a boolean operator such as "equals", "less than", etc. Based on the values on the left and right side and the operator, the expression will evaluate to True or False.

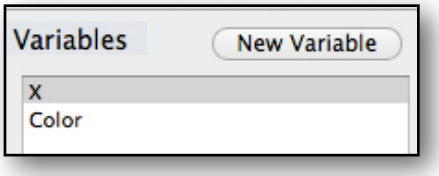
As with List nodes, generally a set of expressions that covers various possible cases will be built and added to the Logic Block as a group.

If any type of variable **other than** a List variable is selected in the Variable list, the expression tab is selected. The expression can be built in the left window and then added to the list of "Nodes to Add" in the right list.

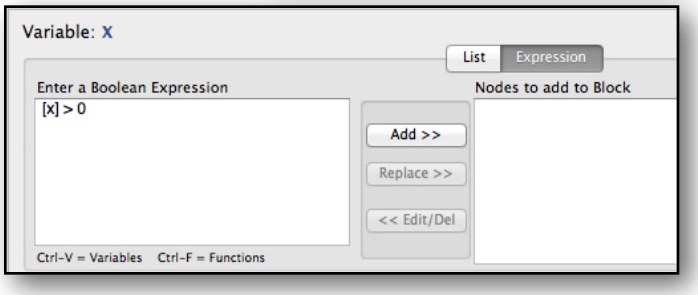
In expressions, Corvid variables are always indicated by the name of the variable in square brackets. (Only List conditions use the name of the variable without square brackets)

To build 2 conditions that test if the value of the numeric variable "X" is greater than 0, or less than or equal to 0:

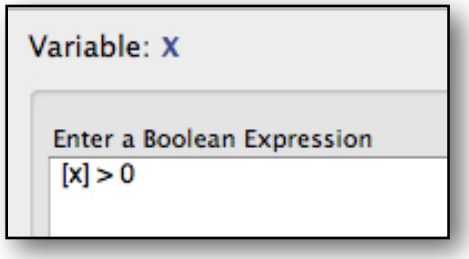
- Click on the numeric variable X in the variable list to select it.



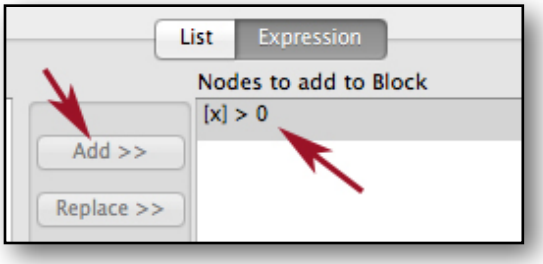
- Since this in not a List variable, the Node Builder will switch to an edit box for expressions.



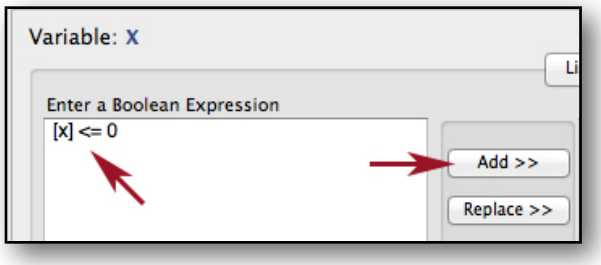
- Enter "[X] > 0" Variable names are NOT case sensitive, but must match the name of the variable exactly in all other ways. A popup list of all variables in the system can be displayed by clicking Control-V. Just double click a variable in the list to add it to the expression.



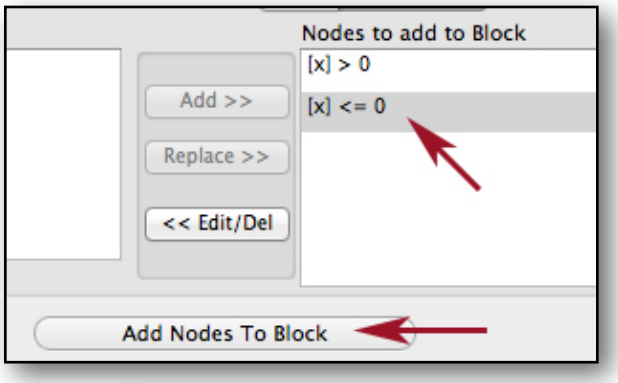
- Click the "Add" button to add this condition to the "Nodes to Add list. (A shortcut is to just press the Return key when the expression is complete to add it to the "Node to Add" list).



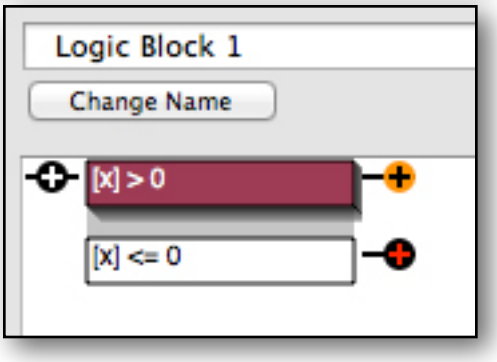
- Now enter the expression “[X] <= 0” and click the “Add” button.



- Since these are the only 2 conditions needed, click the “Add Nodes to Block” button. (A shortcut is to press the Return key when there are nodes in the “Nodes to Add” list and the edit box for adding an expression is empty. This is the same as clicking the “Add Nodes to Block” button.)



- The nodes are added to the block at the Active Insertion Point. The nodes are in the same oder as the conditions in the “Node to Add” list. The Active Insertion Point is moved to the right of the top node added.



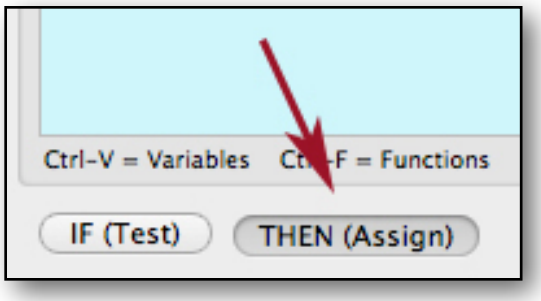
IF Expression Syntax

IF node expressions must be “Boolean expressions”, meaning they evaluate to True or False. The expressions can be as complex as needed. Parenthesis can (and should) be used to make the order of operation clear. There are many functions that can be used in expressions. These are covered in Appendix A.

6.6 THEN Nodes

Building rules also requires adding THEN nodes. These assign a value to a variable. They are built very similarly to IF nodes.

First click on the “THEN (Assign)” button at the bottom of the window to put the node builder in the THEN mode. Notice the background color changes to blue. This is to remind you that you are building THEN nodes.



List Variables

List variables can be used to build THEN nodes are built by simply selecting one or more values that will be assigned.

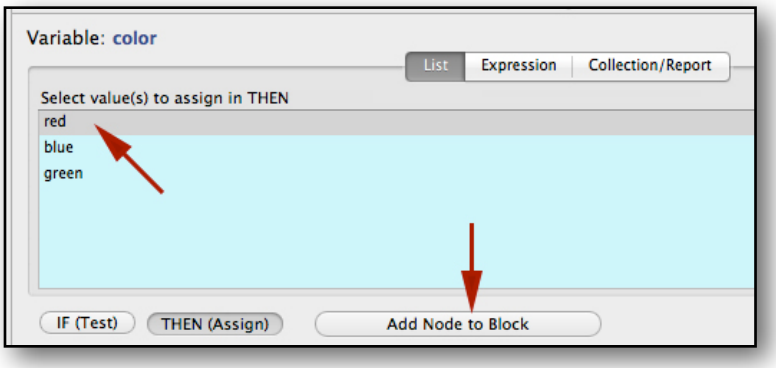
- Click on a list variable in the Variable list to select it
- Its associated value list appears in the blue list box.

Unlike IF node groups, only a single assignment is made in a THEN node - though multiple values can be assigned. (When multiple values are assigned in the THEN, they are combined with AND and all values are assigned if the rule fires).

Select the value or values to assign. It is not necessary (or possible) to build multiple groups as in the IF part.

Click a value(s) to select it and click the “Add Node to Block” button.

This will add the THEN node at the active insertion point. When the THEN node is added to the Logic Block - also highlighted in blue. When there is only a single value to assign, a shortcut is to just double click on the value. This is the same as selecting it and clicking the “Add Node to Block” button.



Other Types of Variables

All variables that are not List variables or Collection variables have a THEN node that assigns a value to that variable. The form is always:

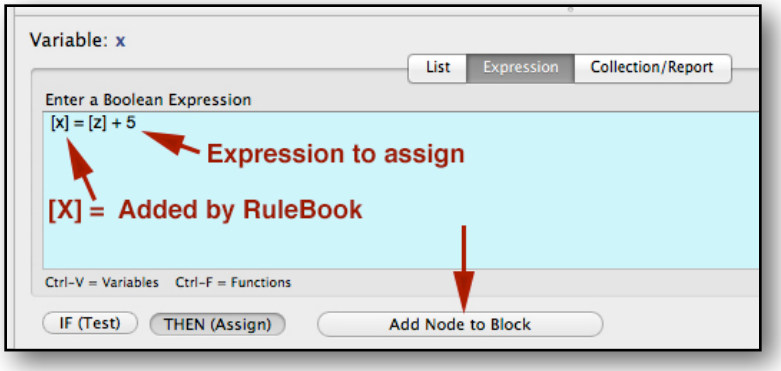
[varname] = expression

The [varname] is the name of the variable in square brackets. The expression is any expression using other variable, functions, etc that evaluates to a value of the correct type.

Corvid will automatically build the “[varname] = “ when you click on a variable in the Variable panel (except a List or Collection variable). The blue edit box shows “[varname] = “. Just enter the value to assign to the variable. This can be a simple value or a complex expression with other variables, functions, parenthesis, etc. **The expression must evaluate to a value that matches the Type of the assignment variable.** For example, a numeric or Confidence variable must be assigned a numeric value. A string variable must be assigned a string value. A date variable must be assigned a date value.

Corvid can display the variable and function list popup windows to help build the expression by pressing Control-V and Control-F. Once the full expression is built, click the “Add Node to Block” button or just press the return key.

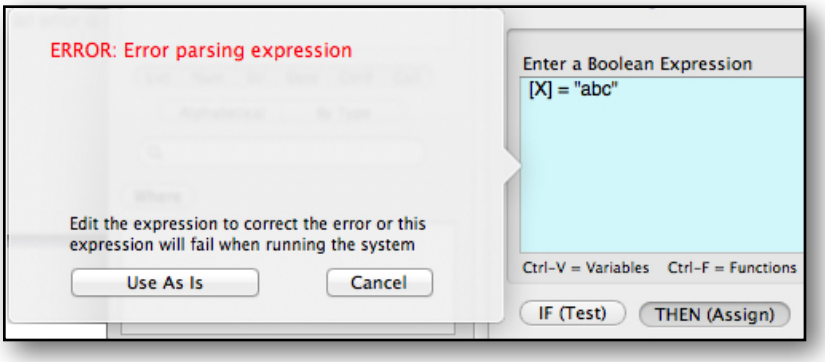
Confidence variables should be assigned a numeric value. This can be a simple value or come from complex expressions. However,



remember that a Confidence variable will combine ALL the values it is assigned based on the confidence mode selected for the variable. Confidence variables are covered in detail in chapter 4.12.

Syntax Checking

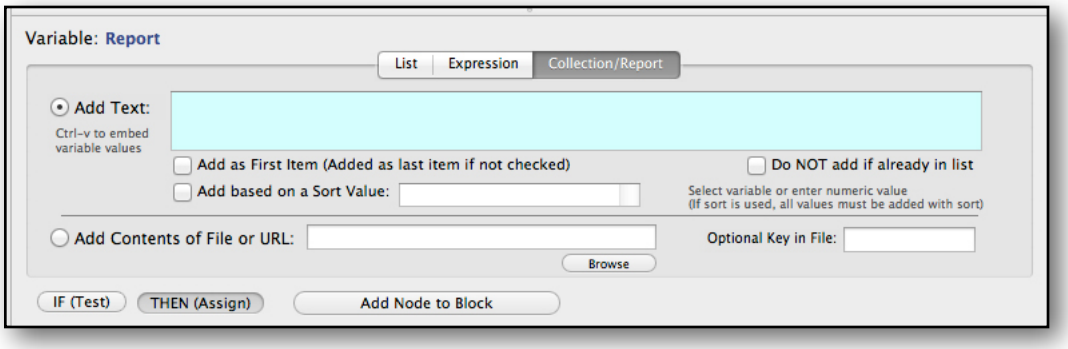
Corvid will check the expression syntax before adding it to the Logic Block. This will check for many types of syntax errors.



Here the numeric variable [X] is assigned a string value. Since a string value cannot be assigned to a numeric, It produces an error message. Sometimes Corvid determines there is an error, but may not precisely know the cause. It could be we meant to have the variable be a string Type and selected the wrong variable, or perhaps the value assigned was supposed to be numeric. When an error is display, modify the expression and try again to add it. It is possible to override the syntax check by clicking the “Use As Is” button, but it should be done with care since if the syntax is incorrect when the system is run it will produce a runtime error that may be more difficult to find.

6.7 Collection Variables

The last type of THEN condition that can be added is one for Collection variables. Collection variables are generally used to build reports or lists of items. A Collection variable’s value is a list of strings. It can be thought of like a shopping list with various rules adding different items, but actually it is more likely to be a HTML page or report. Collections are very useful for many types of advanced systems. Collection variables have many options, but basically text can be added to the list of strings that makes up the Collection variable’s value. To add an item, enter it in the blue edit box.



By default, it will be added at the end of the list. It can be added as the first item in the list by checking the “Add as First Item” check box. You can also have Corvid not add it to the list if it is already in the list by checking the “Do not add if already in list” check box.

Items can be added with a sort value, so that the one added with the highest sort value will be at the top. There are various properties for Collection variables to work with sorted lists.

Alternatively, a Collection variable can get its value list from a file or URL. This can be a report template that may structure an HTML page that includes the value of various variables embedded in it.

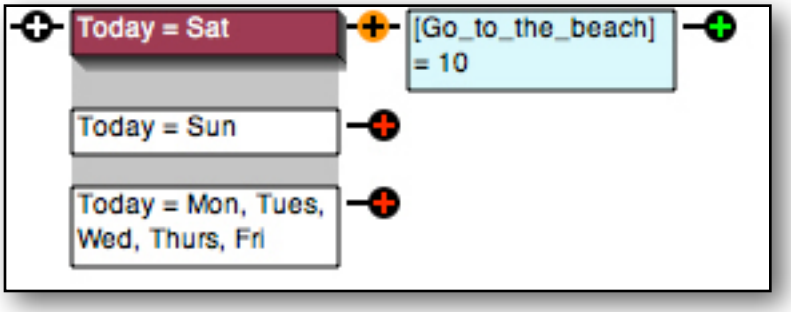
Using Collection variables is covered in detail in chapter 15 on building reports.

6.8 Inserting New Nodes

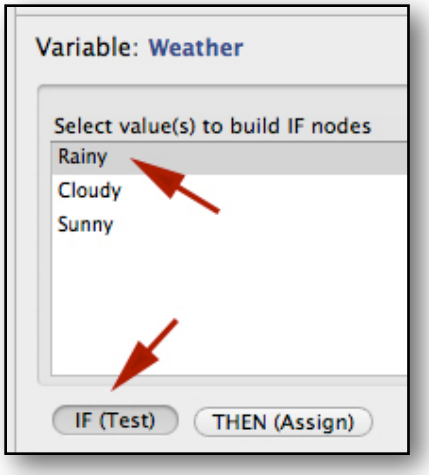
New nodes and groups of nodes can be inserted at any insertion point. Just click on an insertion point to select it. Depending on the insertion point's surrounding nodes, it may be possible to insert an IF node, THEN node or either. Corvid will automatically enable or disable the IF and THEN buttons at the bottom of the window and will switch to either IF or THEN mode when only one type of node is allowed. (THEN nodes cannot be inserted between IF nodes and IF nodes cannot be inserted between THEN nodes).

Build a node or group of nodes as when first building the Logic Block. The only thing to remember is that when a group of associated IF nodes is added between nodes, **the TOP node in the group will be put between the existing nodes** and the other nodes will create new branches in the tree. Select the IF node that should be added to the rule being edited by using the arrow buttons to make it the top item in the "Nodes to Add" list.

For example, suppose there is a Logic Block that says the user should always go to the beach is it is Saturday, and you decide only to give this advice if the weather is sunny. This can be fix by adding an IF condition "The weather is sunny". between the "Today = Sat" and "Go to the beach" nodes.

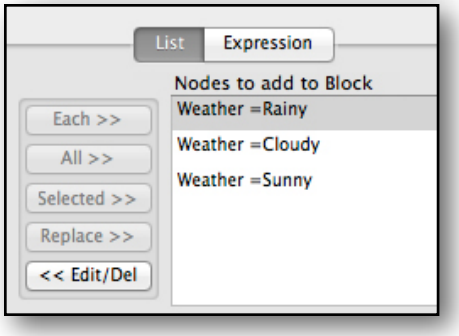


Click the insertion point where the new node will be added. This will be displayed as the orange "Active Insertion Point". Since this is between an IF and a THEN node, either an IF or THEN node can be added. An IF node is wanted, so make sure the "IF Node" button is selected and the "Weather" variable selected in the Variables panel.

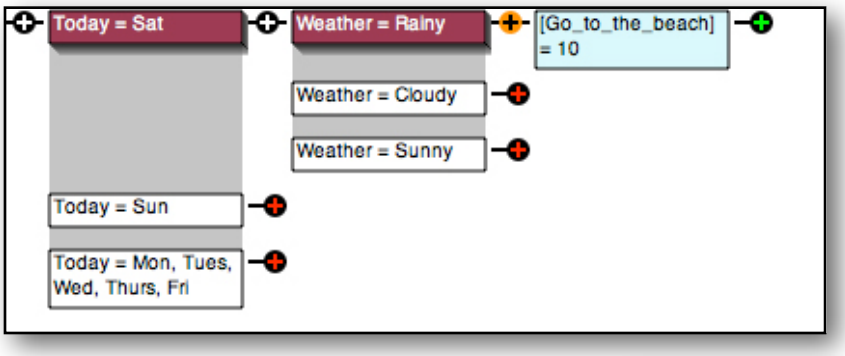


The "Weather" variable is a List variable with 3 values. If this was the last node on the branch, the values could be used to build conditions in any order. But here it is used between nodes and the TOP condition in the nodes to add list is the one that will be inserted between the existing nodes.

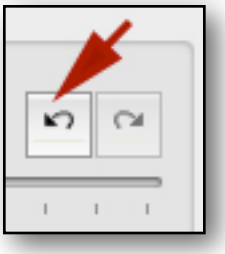
If the "Each" button was clicked to add the nodes in their normal order, the "Weather = Rainy" node would be at the top.



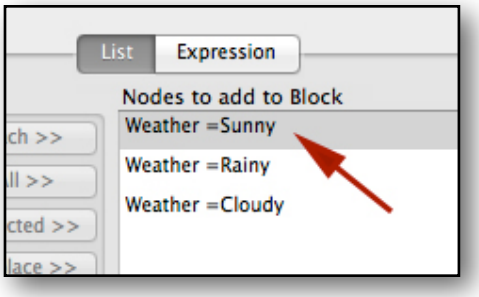
Adding this group of nodes would result in a Logic Block that recommends going to beach only in the rain - exactly the opposite of what we intended.



Errors like this can be easily fixed by clicking the Undo button in the upper right.

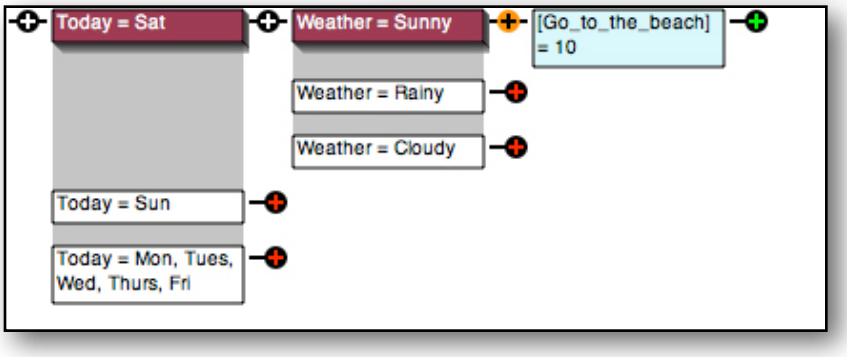


Now build the nodes with the “Sunny” value as the top node. This can be done by adding “Sunny” first, or clicking “Each” and then moving “Sunny” to the top with the node arrow keys.



Adding these nodes to the Logic Block produces what was intended.

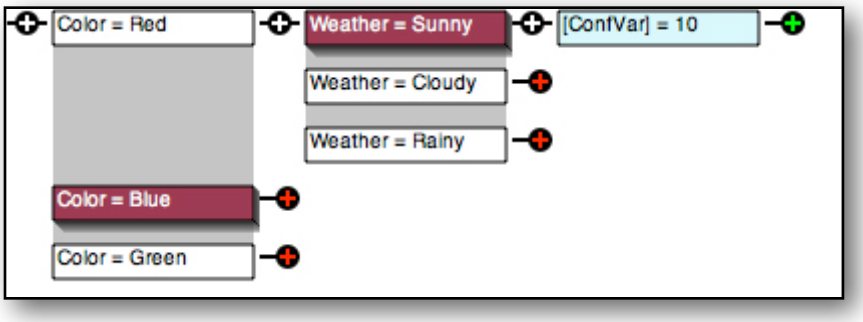
Notice that the order of the other nodes does not really matter since they will end up as the end of a branch and can be expanded as desired. It is only the top node that is inserted between nodes and matters.



6.9 Editing Logic Blocks

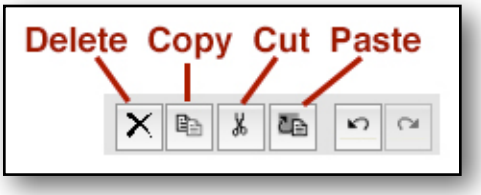
Logic Blocks can be edited in various ways to add the rules or modify the logic. In most respects this is done the same as many other programs - cut, copy, paste, delete and modify. However, since the tree structure of the Logic Block must be maintained, some of these actions have special limits or behaviors to maintain the structure of the tree.

Individual nodes and node groups in the Logic Block can be selected by either:



- Clicking on a node.
- Holding down the Shift key and clicking on multiple nodes to select them all.
- Clicking on an insertion point to select all the nodes that lead up to that point.
- Command clicking on an insertion point to select he tree to the **right** of the insertion point.

Once nodes are selected the Delete, Cut and Copy buttons can be used. Once one or more nodes have been Cut or Copied, they can be pasted.

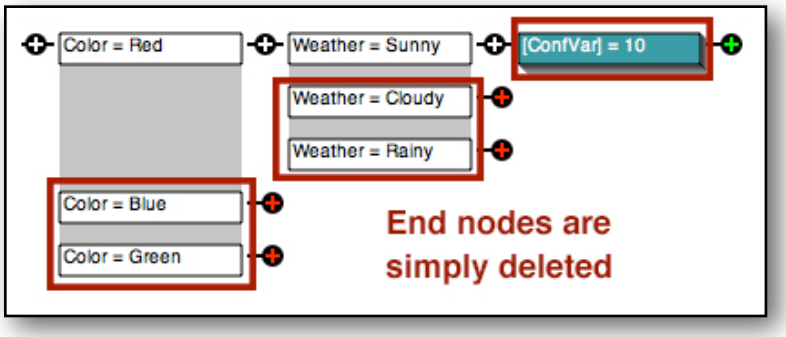


Note: Cut, copy and paste for nodes is controlled ONLY by the buttons in the Logic Block window. The normal Mac Command-C, Command-X, Command-V controls only apply cut, copy and paste for text being edited in a text box. They do not apply to nodes.

Delete

To delete a node(s) in the Logic Block, select the node(s) and click the Delete button at the top right of the Logic Block.

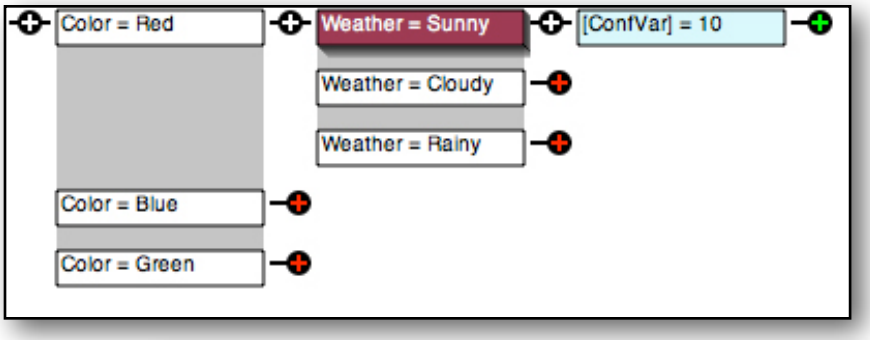
If the node is the last node on a branch it will just be deleted.



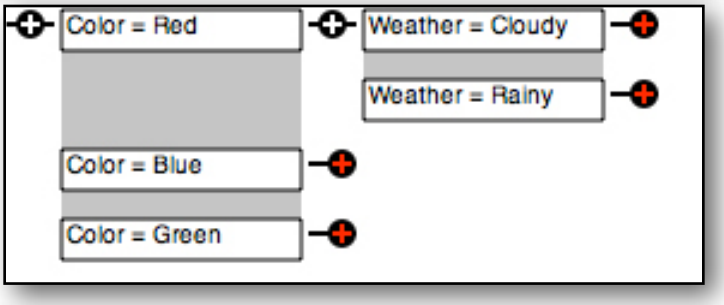
If the node is a single node (not part of a group of related nodes) with nodes to the right of it, it will be deleted and the node(s) to the right will move left to fill the gap.

However, when a node to be deleted is part of a group of related nodes with a gray background, Corvid cannot simply delete it and move the nodes to the left. This would break the group and the tree structure of the Logic Block. Because of this, if a node in a group of related nodes is deleted, the node and **ALL NODES TO THE RIGHT OF IT** will be deleted and the next node in the group will be moved up.

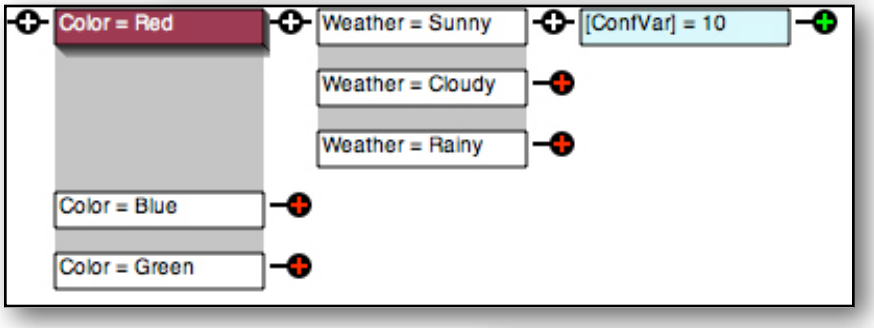
For example: The "Weather=Sunny" node is part of a related group of nodes. If it is deleted, the THEN node to the right cannot simply be moved left. Instead the selected nodes and all nodes to the right are deleted and the remaining nodes in the group move up to fill the gap.



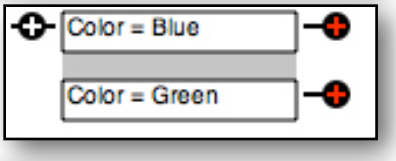
This may seems like odd behavior, but it is necessary to maintain the tree structure.



If the "Color=Red" node was deleted, it would delete all the nodes to the right of it. This includes the full "Weather" group.

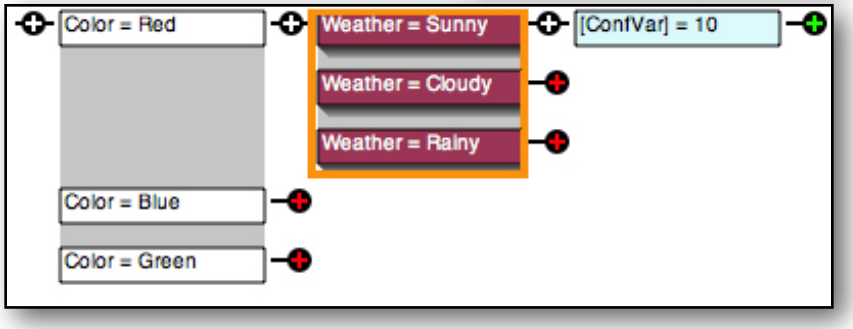


The nodes beneath it would move up to become just:

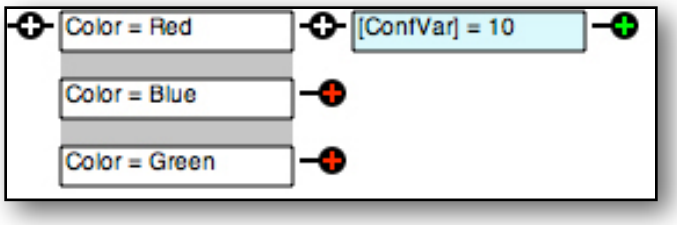


Deleting nodes in the middle of the block needs to be done with some care to not inadvertently delete more than was intended. However, if Delete takes away more than expected, just click the **Undo** button to bring it back.

Entire groups of nodes can also be deleted. To select a group of related nodes, double click in the group. An orange border will be displayed around the group indicating that the full group is selected.



Now click the Delete button and the group will be deleted. The nodes to the right of the **TOP** node in the group will be moved to the left. **The nodes to the right of any other nodes in the group will be deleted.**



Multiple nodes can be selected and deleted at the same time. Use a Shift-click to select multiple nodes in the block and then click delete. Each node will be deleted from the bottom row up based on the rules for that node, which may lead to nodes to the right also being deleted.

Cut and Copy

Nodes can be cut, copied and pasted into the Logic Block. However, the nodes that can be pasted into a block are limited to those that will still retain the Logic Block structure. Corvid will disable the cut and copy buttons if the set of nodes selected cannot be pasted into a block.

The main sets of nodes that can be copied and pasted are:

- Individual nodes (click on the node).
- Sequential nodes on the same branch (Shift click on the nodes).
- A group of associated nodes (double click to select).
- The tree up to an insertion point (Click on an insertion point).
- The tree to the right of an insertion point. (Command click on an insertion point).

Once an allowed set of nodes is selected, click the Copy button. This will copy the selected nodes to the paste buffer without changing the Logic Block.

Alternatively, the nodes can be cut, which is equivalent to a Copy followed by a Delete.

This can be done by clicking the Cut button.

Cut will copy the selected set of nodes to the paste buffer, but the subsequent delete will follow the same rules as a normal delete and **may result in additional nodes to the right of the selected nodes also being deleted**. These additional nodes will NOT be in the Copy buffer.

Paste

To paste content into a Logic Block:

Click on an insertion point to indicate where the content should be pasted, then click the Paste button.

Corvid checks paste actions to prevent damage to the tree structure. Pastes must maintain the IF/THEN structure, and THEN nodes cannot be pasted between IF nodes and IF nodes cannot be pasted between THEN nodes. Attempting to paste content that would damage the tree structure will result in an error message.

If a group of related IF nodes is pasted into the tree, the TOP node will be added to the branch with the selected insertion point.

As with all editing actions, if the Paste does not produce the structure that was intended, click the Undo button and try a different approach.

Editing Node Groups

A group of related nodes marked with a gray background can be edited by double clicking on any of the nodes in the group - note this is clicking on the nodes, not the insertion points. The node group will be highlighted as selected and marked with an orange border. The individual nodes from the group will appear in the “Nodes to Add” list and can be edited or replaced.

Nodes can be edited to change the order by clicking on the arrow keys, or can be replaced by new nodes built by clicking in the Variables panel and building new groups of nodes. When new nodes are created and the “Replace Nodes in Block” button clicked, the first new node will replace the old first node and any nodes to the right of that node will remain **unchanged**. This applies to all the new nodes, so when replacing nodes in the middle of a Logic Block it is important to remember that the other nodes in the block will be unchanged. **If existing nodes in a group are reordered, this will NOT change the order of the nodes to their right.**

Nodes built with other than list variables can be edited the same way. Just double click on one of the nodes in the group and the nodes will appear in the Node to Add list.

Either:

- Select a node and click the Edit /Del button to move the content to the left edit box.
- Make whatever changes are needed in the expression.
- Click the Add button or press the Return key, to add it back into the list.

OR

- Enter a new expression in the left edit box.
- Select the node to replace on the right and click the Replace button.

Nodes can also be added by entering a new expression on the right and clicking the Add button.

When building list nodes, the nodes in the group should all be based on the same variable. When building expression nodes, any variables can be used. There is no requirement that the nodes in the group form a complete set, or even be related, but they will all have the same parent nodes when building a rule.

If there are more new nodes than there were old nodes, the extra nodes will be added to the block but will not have any nodes to their right. If there are less new nodes added to the block than existing nodes, the extra nodes **and all nodes to their right** will be deleted from the block.

If when editing nodes, an error is made, just click the Undo button to move back to the earlier state. This can be done multiple times to step back through the sequence of changes to the block. If you step back too far, click the Redo button to return to the state just before the last undo.

7 Backward Chaining

7.1 Introduction

Understanding backward chaining is essential to fully utilizing the power of the Exsys Inference Engine and building efficient, well structured expert systems in Corvid.

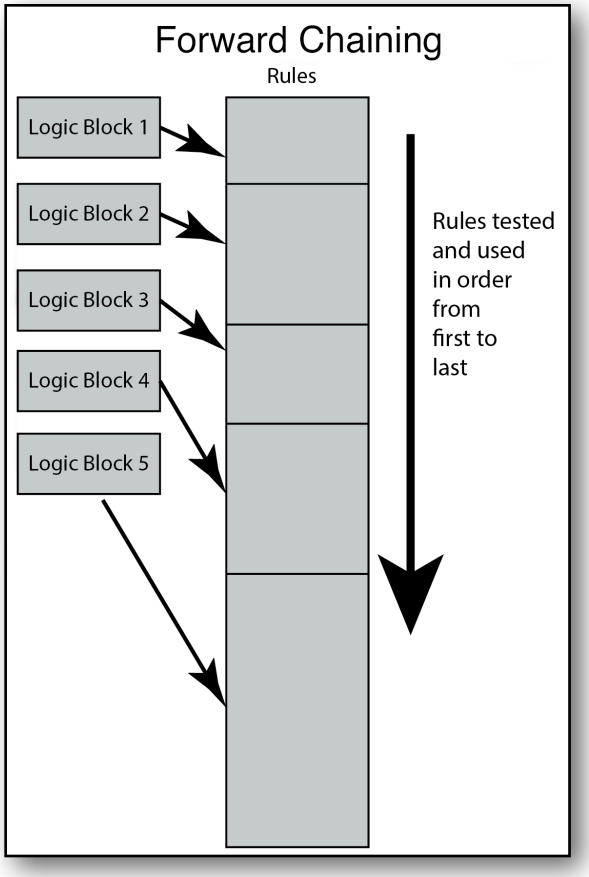
Backward chaining is probably the most important concept to master in using Corvid. It is what separates running rules with an Inference Engine from other types of programming. Backward chaining enables solutions to complex decision making problems to be implemented quickly and maintainably.

Backward chaining may seem strange since it is relatively unique to expert systems. It is a very important concept to fully understand. It can seem confusing at first, but once the principles behind it are clear, it is easy to use.

Basically, what backward chaining means is that whenever the system logic needs the value of a variable, it will search through the rules to see if there are any rules that might assign a value to the variable, and will automatically invoke those rules to derive the needed value. This can be repeated recursively many levels deep.

Simply put, if the system is testing an IF condition that calls for the value of a variable, and there is another rule anywhere in the system that **sets** that same variable in the THEN part, that rule will be immediately used to try to derive the value for the variable.

To link rules together use the same variable in the IF part of one, and the THEN part of another. All linkages are implicit and dynamic. This is very different from traditional programming where if a variable is needed at a particular point, there would have to have been explicit code to derive the needed value. When derivations are many levels deep or dynamic, the traditional programming approach quickly gets very complicated and difficult to maintain.



7.2 Forward Chaining

Part of the problem in understanding backward or forward chaining, is the terminology. There is nothing really “backward” about backward chaining or “forward” about forward chaining. The terms have a long history going back to the early days of expert systems in the 1980s. Unfortunately, despite being somewhat confusing, the terms have stuck and are the ones commonly used.

Forward chaining is based on testing rules in sequential order. Each row in a Logic Block builds a rule and these are tested in order from the top row down. The rules from the first Logic Block are first, then the rules from the next Logic Block, etc. When running in **Forward Chaining**, the rules are tested in order - first rule first, then the next, etc.

When the IF conditions in a rule are true, the rule “fires” and the assignments in the THEN part made. Then the next rule is tested, then the next, etc.

As rules are tested in forward chaining, they may require asking the user for variable values to determine if the IF tests are true, and rules that “fire” may set the values for other variables. The forward chaining, “Order based” approach is much more like traditional programming since the Inference Engine just steps through the If/Then rules in order and fires them when the IF conditions are true. It is quite easy to know what rule will be used next and when particular variables will be needed. This makes the forward chaining “Order based” approach relatively easy to understand and visualize. This is particularly true of developers with a programming background who expect their “code” to be executed in order. Many systems can be built this way, but it does not make use of the real power of the Inference Engine.

Forward Chaining Limitations

Forward chaining is a very procedural and order dependent way to approach a problem in steps. It is an approach that can often be implemented without any “inference engine”.

Many procedural instructions are described in steps - first step first, second step next, etc. Corvid supports Forward Chaining and there are vast numbers of procedural operations that can be converted to forward chaining systems in Corvid - from baking cookies to fixing jet engines. Corvid’s forward chaining option works well for converting these procedural operations to online systems.

However, Corvid systems that capture decision-making knowledge of how an expert solves a complex multilayer problem often cannot be described using defined sequential steps.

Corvid is based on If/Then rules that describe aspects of making a decision. These are built, arranged and structured in Logic Blocks, but here we will just look at the rules themselves regardless of how they are built.

Corvid rules are made up of IF conditions (boolean expressions) using variables that evaluate to TRUE or FALSE.

When a rule has multiple IF conditions, they all must be TRUE for the rule to be TRUE.

When a rule is TRUE, its THEN conditions will assign a value to a variable(s).

Suppose there is a system to diagnose various possible problems with a machine. Two of the rules in the system (though there would also be many more) are:

IF

    Machine is running = No

AND [TEMP] > 150

THEN

    The problem is the high temperature shutoff

IF

    The temperature warning light = ON

THEN

    [TEMP] = 150

f these rules are run in order with **forward chaining**, the first rule would be tested first. The system would ask the end user if “the machine is running” since that is the first IF condition. If that value was “No”, it would then ask for the value of [TEMP] to test the second condition. But that is a problem, since the second rule can set the value for [TEMP], the system should use it, and we certainly don’t want to ask the user for a [TEMP] value and then change it in the next rule. It would be better to use the second rule to derive the value for [TEMP] instead of asking the user.

One solution would be to move the second rule to be first in the list:

IF

The temperature warning light = ON

THEN

[TEMP] = 150

IF

Machine is running = No

AND [TEMP] > 150

THEN

The problem is the high temperature shutoff

Now the rule to set the value for [TEMP] occurs first. The user will be asked about “The temperature warning light“ and if the value for [TEMP] is set, it can then be used in the second rule. But this is a problem since it only makes sense to derive the value of [TEMP] when the “Machine is running = No” condition is true. If the machine is running, [TEMP] is not needed is not needed (at least not here) since the second rule is automatically false because its first condition is false, and ALL the IF conditions must be true for the rule to be true. In that case, either asking or deriving [TEMP] is not needed, so we do not want to start off by asking about “The temperature warning light“, as would happen from the reordering.

One solution is to change the first rule to:

IF

Machine is running = No

AND The temperature warning light = ON

THEN

[TEMP] = 150

Now [TEMP] will only be derived if the machine is not running. That is not the same rule as before. There may be other rules in the system that need [TEMP] even though the machine is running. Also, when there are many things being checked, explicitly linking and setting restrictions on the rules this way can get very complicated and makes systems difficult to build and maintain.

No matter how they are arranged, the 2 rules just do not work ideally as a forward chaining system - and that is just with 2 rules. A real system may have hundreds of rules with many internal interrelationships. Creating a rule order that can be run sequentially may be impossible. Forcing an order for procedural reasons and adding extra conditions may make the rules difficult to understand and maintain, or repetitive.

Also human experts often make decisions based on “high level” logic that applies, but which has the details and specifics provided by other lower level logic. For example,

IF

The customer has high risk tolerance

AND: Meeting objectives requires rapid growth

THEN

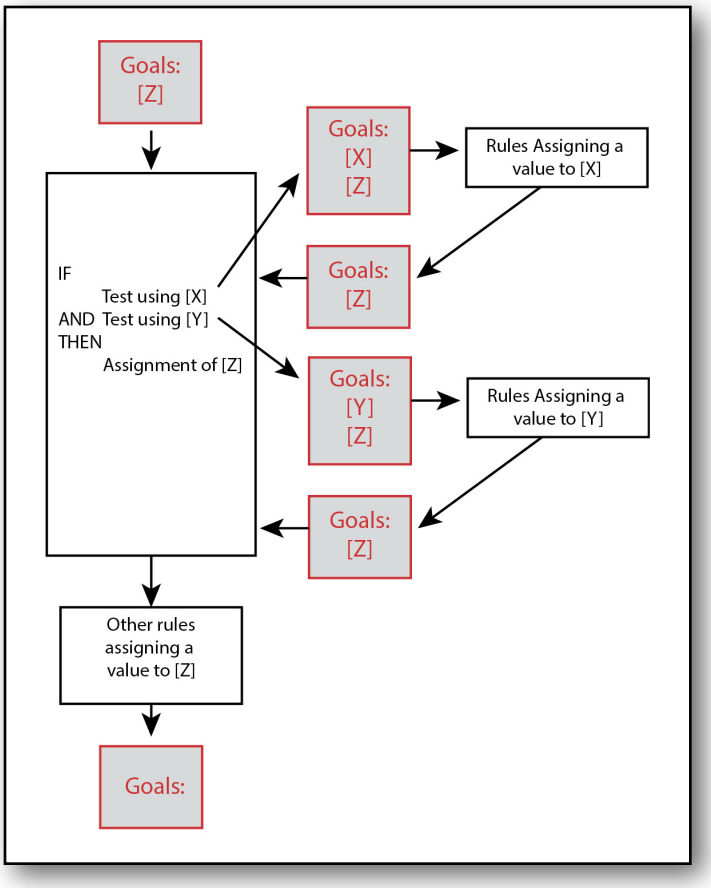
Mutual Fund X is a good choice

This is high level logic, but the system should not directly ask the user about their “risk tolerance” or “objectives” since the end user may not be able to reliably answer those questions. Instead the system should derive these values from other rules, but now there are the same internal interrelationship issues as with the 2 rule sample above. What questions should be asked to derive values, when are they needed, when are they not appropriate, how should the rules be arranged. It can get very complicated very quickly. Fortunately backward chaining solves this problem and makes it very easy to handle.

7.3 Backward Chaining

Backward Chaining is also referred to as “Goal Driven”, which is a much better name. Rather than running the rules in sequential order, the Inference Engine is given a “Goal” to achieve. **In Corvid, a “Goal” is always determining the value for a variable.** The Inference Engine examines the full set of rules to determine which rules can assist it in achieving the current goal by assigning a value to the goal variable.

The only rules that matter, at that moment, are those that can potentially set a value for the goal variable by assigning a value to the goal variable in their THEN part. **These are the only rules relevant to the immediate “Goal” and all other rules are, for the moment, irrelevant to the “Goal” and are temporarily ignored.**



The Inference Engine will only test the currently relevant rules, even though this may mean skipping over many “irrelevant” rules. **The Inference Engine will test and use rules based on its need to achieve “Goals” rather than the order that the rules were written in.**

As with Forward Chaining, in Backward Chaining the Logic Blocks in a system create a list of rules. However, the Inference Engine uses those rules to set the value for a “Goal” variable. Once a Goal variable is set, the Inference Engine tests and uses ONLY the rules that set a value for the Goal variable in their THEN assignments. These are the rules that are “Relevant” to the immediate Goal. Other rules that do not set a value for the Goal variable are temporarily ignored as irrelevant, regardless of rule order.

Testing a “Relevant” rule often requires getting the value of another variable that is used in that rule’s IF conditions. When that happens, the needed variable becomes the new Top Level goal, pushing down the other Goals in the list. The Goal variables can be thought of as a list where new Goals are added to the top of the list and push down the other Goals already on the list. Those other Goals are not forgotten, but are temporarily superseded as the Goal being worked on by the Inference Engine.

Whenever the Top Level goal changes, the rules that are “Relevant” to that Goal change, and the Inference Engine immediately changes to focus on the new Relevant rules. When a Goal variable has its value set, it is dropped off the top of the Goal list and the next Goal down again becomes the new top level goal - causing the Inference Engine to switch focus to trying to again set a value for that variable, but this time with more data to use. This continues until the Goal list is empty.

Programmers that are used to procedural approaches to writing systems may at first find Backward Chaining confusing and unexpected. This is because with backward chaining, the Inference Engine does much of the work rather than depending on the programmer to precisely set the sequential rule order. Once you get use to it, it really makes building systems MUCH easier.

Looking at the example from above:

IF

Machine is running = No

AND [TEMP] > 150

THEN

The problem is the high temperature shutoff

IF

The temperature warning light = ON

THEN

[TEMP] = 150

With Forward Chaining, there was no way to order the rules that ran correctly. With backward chaining, we use “Goals” and do not care about the rule order. Instead of procedurally running the rules in order, with backward chaining we set the goal to be “Determine what the problem is with the machine”. In a real system there may be many rules that identify various possible problems. In the 2 rules above, the Goal would be determining if “The problem is the high temperature shutoff”. (How a Goal is set will be explained shortly, here we just assume that has been done.)

This tells the Inference Engine that only rules that can set a value for “The problem is the high temperature shutoff” are relevant to the immediate Goal and need to be tested.

This happens to be the first rule, but the same rule could be anywhere in a system and would work the same way. To determine if this rule is true, the Inference Engine tests the first IF condition and needs the value of “Machine is running”, which will have a value of “Yes” or “No”. This is just a Corvid List variable.

Since this variable is required for setting the current Goal, it becomes the new top level Goal. Now “Machine is running” becomes the top level goal briefly replacing “The problem is the high temperature shutoff”. The “The problem is the high temperature shutoff” goal is not forgotten, just temporarily replaced as the top level goal by being pushed down in the Goal list. Remember the Inference Engine is always trying to set a value for the **Top Level** goal - even though that will be constantly changing. Since there is

no rule that can set the value for “Machine is running”, the end user is asked to input the value. That satisfies the “Machine is running” goal and it is removed as the top level goal. The “The problem is the high temperature shutoff” goal, which had been temporarily superseded and pushed down in the Goal list, pops up again becoming the top level goal.

If the end user input that “Machine is running = No”, the rule first is again potentially a way to set a value for the “The problem is the high temperature shutoff” goal. The first IF condition in the rule is true so now the Inference Engine needs to test the second IF condition and requires the value of [TEMP]. As before, [TEMP] now temporarily becomes the Top Level goal pushing the other Goal down in the list. However, this time there is a rule that can set a value for [TEMP]. Since [TEMP] is now the Top Level goal, the Inference Engine gives it priority and immediately switches to testing any rule that sets [TEMP]. There is a rule that sets [TEMP] so it’s IF conditions now need to be tested and the system now needs the value of “The temperature warning light” - which becomes the new Top Level Goal, superseding and pushing down the current goals. At this point the goal list is:

The temperature warning light

[TEMP]

The problem is the high temperature shutoff

Since there are no rules that can set “The temperature warning light”, it is asked of the end user and dropped off the goal list, bringing [TEMP] back to the top of the Goal list. If the end user said that “The temperature warning light = ON”, the second rule will have set a value for [TEMP], and [TEMP] can also be dropped off the goal list, bringing “The problem is the high temperature shutoff” back to the top. Now the first rule again becomes “relevant” and (with the appropriate user input) can fire assigning a value to “The problem is the high temperature shutoff” and it can be dropped off the goal list. Since the goal list is then empty, the Inference Engine is done. Since there are no remaining Goals, any other rules that might be in the system would be ignored.

This may seem complicated, but all the complexity was handled by the Inference Engine and happened invisibly to the developer. The developer just provided the heuristic rules, in any order, and instructed the Inference Engine to use them.

If the end user had said “Machine is running = YES”, the first rule’s first IF condition would have been false, and the first rule would be false and dropped. In that case, [TEMP] would **not** have become a goal and the second rule would never have been needed or tested. The “temperature warning light” question will only be asked if it is actually needed. If the rules had been in a different order, the system would have run the same way. The second rule did not need any extra condition to control how it was used, since it will ONLY be used if it assigns a value to the Goal variable. If [TEMP] is not the Goal, the second rule is irrelevant - however, if **ANY** rule in the system becomes relevant for **ANY** goal and needs [TEMP], the second rule will be used to derive the value or [TEMP]. All problems with forward chaining are gone automatically just by running the rules with backward chaining.

Best of all, even if there were many rules with many levels of interdependencies, it would all still automatically work correctly.

For complex or multilevel logic, backward chaining has many advantages over forward chaining, since the “Goal” changes dynamically and automatically. The adding and removal of goal variables from the Goal stack happens dynamically, and occurs many times when running a backward chaining system. The rules that are relevant to the **immediate** goal constantly are changing and rules are used in an order that may be very different from the way they are ordered in a system. This can be confusing at first since it is not following the procedural order of the rules - but it is actually following a very logical, structured and focused approach to using the rules in the system that allows creating and maintaining much more complex systems that ask only relevant questions based on need for the data.

Forward Chaining	Order Driven	Rules used in sequential order
Backward Chaining	Goal Driven	Rules used based on relevance to “Goals”, which change dynamically

Backward chaining allows building small sub-sets of rules that form reusable “modules” to handle specific portions of a problem. The variables set in these modules can then be used in higher level rules that describe a more general, abstract solution to the overall problem. Backward chaining will automatically use the modules as needed by the higher level rules. The Inference Engine will automatically find the “relevant” rules at each step of the process. If the logic in a module changes, all this is needed is to change the rules in the module and the changes will automatically apply everywhere the module is used.

This can be confusing at first, but is actually simple. In Backward Chaining, the Inference Engine uses the rules the way a person would think about how to solve a problem - high level logic that uses other lower level rules to provide the values needed in the higher level rules. People don’t consciously think about “goals”, but if an expert is explaining how they made a decision, and they say “Because of X and Y, I knew...” and you ask them “How did you know X?”, They will explain lower level logic that let them know X. In Corvid terms, X was briefly their top level “Goal” and those rules set X. When the rules are structured in Corvid, the goals become more apparent, but they are actually the same way experts solve most complex problems.

7.4 The Big “To Do” List

Backward Chaining is all based on a dynamic Goal List. One way to think of this list is as a “To Do” list made up of a stack of sticky notes. Items can be added to the top of the stack, and when they are “done” they get peeled off so that the next item in the list again becomes the focus of the Inference Engine.

In Corvid, the “Goals” in the “To Do” stack are always Corvid variables.

The Inference Engine is always **completely** focused on assigning the value to the variable that is on the top of the goal list “To Do” stack.

Whenever the goal (Variable) on the top of the “To Do” stack changes due to adding or removing a goal, the Inference Engine **immediately** switches whatever it was doing to trying to assign the value to the new top Goal variable.

To assign the value to the goal variable, the Inference Engine looks through all the rules in the system to find any that would assign a value to that variable in the THEN part of the rule. Those rules become the “relevant” rules and are the only ones tested for that goal variable.

If testing a relevant rule requires the value of any other variable either to test an IF condition or, when a rule fires, to make an assignment, **and that variable does not already have its final value assigned,** that other variable is put on the goal list as the top item in the “To Do” stack. Determining the value of that new goal variable, immediately becomes the focus of the Inference Engine. Processing of all other rules is temporarily suspended, and only the rules “Relevant” to the new variable are tested.

If a “Relevant” rule for the top goal variable is found to be TRUE, the value will be assigned to the goal variable, **and any other THEN assignments in that rule will also be made**, even though they do not involve the goal variable. If any of the assignments require the value of other variables that do not have a value assigned, those will be put on the top of the Goal list and become the focus of the Inference Engine.

If all the “Relevant” rules for the top goal variable are found to be FALSE, and a value cannot be derived, List, Numeric, String and Date variables will be asked of the end user via the user interface. Confidence variables will have a value of 0 and Collection variables will not have had any items added to their value list.

When a goal variable is assigned a value, either by all its “relevant” rules or asking the user, it will be removed from the goal list “To Do” stack and the next variable down in the stack will become the top goal. Since the top goal will have changed, the Inference Engine will immediately focus completely on assigning a value to that variable. Since any other variables that were temporarily above it in the goal list will now have a value, it will automatically return to the same point in the rules but with additional variable values to use. This may now lead to being able to assign a value to the variable, or may lead to other variables being added as the top level goals.

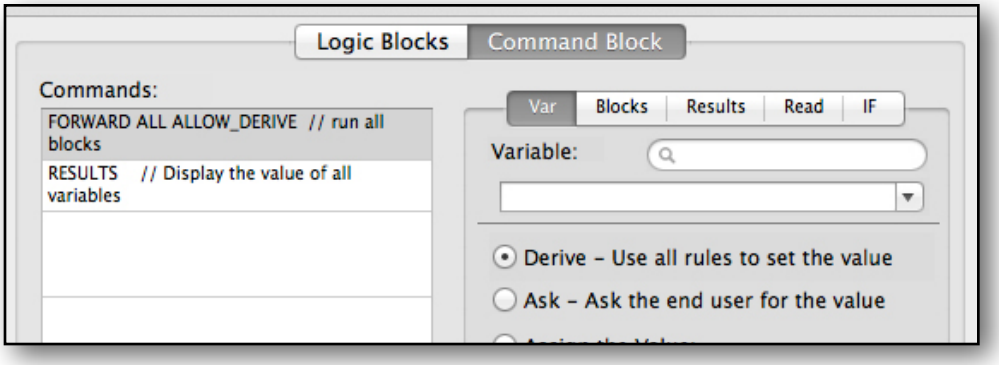
The Goal List “To Do” stack is constantly changing. Since variables are added based on the immediate need for the variable’s value to achieve the lower level goals, the rules are used in a very focused way, with all questions asked of the end user based on the need to get values to meet top level goals. Irrelevant variables are automatically blocked and not asked or derived.

7.5 Starting Backward Chaining

Backward Chaining requires defining one or more Goals by putting them on the Goal stack and telling the Inference Engine to do whatever is needed to get their value(s). Once Backward Chaining is started, Goals are automatically added and removed from the Goal stack as needed and it continues automatically until the Goal Stack is empty.

The procedural running of a system is controlled by the Command Block which is a list of commands that control the Inference Engine. Command Block commands and how to build them is covered in detail in the next chapter, however there are 2 main commands that relate to backward chaining.

Click on the Command Block tab to see the commands in the system.



When a new Corvid system is started, it is automatically given a default command block that can run any system, though virtually all systems will use a more customized command block. (See Chapter 11 on Command Blocks)

The first command is:

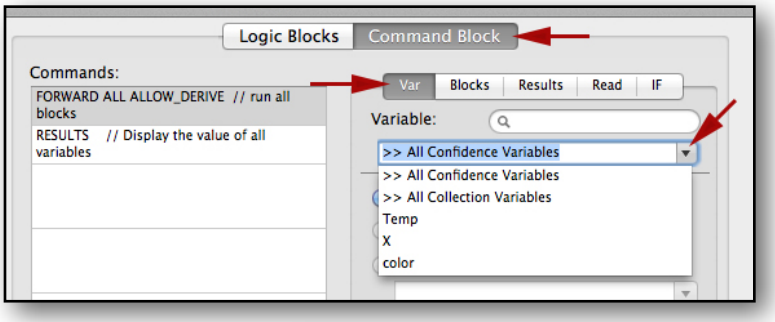
FORWARD ALL ALLOW_DERIVE

This is a special hybrid of both forward and backward chaining that can run any system, though for systems designed for backward chaining,it is usually not the best command to use. FORWARD ALL runs all the rules in a system in Forward chaining - that is the rules are tested and used in the order that they were added to the system. The ALLOW_DERIVE option tells the Inference Engine to use backward chaining to derive any variables that are needed while testing the rules. This command runs all the rules in a system in order, but will still use backward chaining to derive needed values when appropriate by making those variables Goal variables for Backward Chaining.

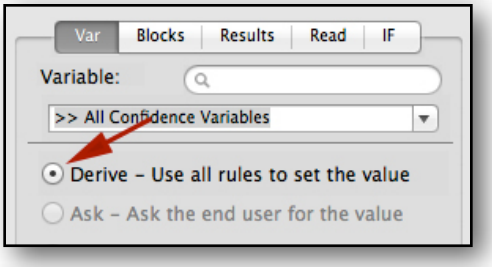
It is a way to use backward chaining when needed, but still give the developer control on the order that rules are used in.

The Command Block command that is generally better for systems designed for pure Backward Chaining is the DERIVE command. This command is used with a variable or a group of variables (by Type) to put those variables on the Goal List and tell the Inference Engine to get their value.

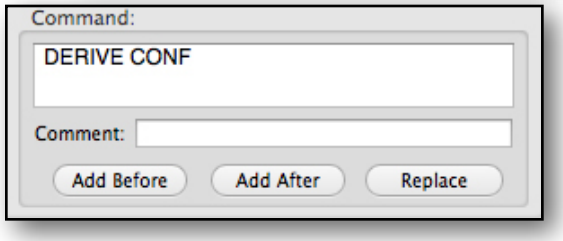
This command is created in the Command Block tab with the “Var” tab selected. The drop down list at the top allows selecting a specific variable in the system or selecting a “All Confidence Variables” or “All Collection Variables” to select all variables of that Type.



Then click the “Derive” radio button. This builds the main command for setting the Top Level goal and starting backward chaining to derive the value.



Then insert the command in the Logic Block using the “Add” or “Replace” buttons to add it relative to the currently selected command.



(See Chapter 11 for details on building and editing Command Blocks)

The Inference Engine will use all the rules in the system, regardless of rule order, to set the values for the selected variable(s) as the top level Goal variable(s).

For example, a system may use a set of Confidence variables for the various possible problems with a machine. These Confidence variables may be set by many rules created by various Logic Blocks that cover the possible problems. Using a DERIVE CONF command will tell the Inference Engine to put each Confidence variable on the Goal List and use backward chaining to derive the value. This approach allows the rules to be very focused way since they will be used “as needed” rather than in the overall rule order.

7.6 Double Square Brackets

Any Corvid string can have the value of any variable embedded in it by entering the name of the variable in **double** square brackets. This can also use variable properties.

Embedded variables will immediately invoke backward chaining to get the value of the variable.

For example, a string added to a report in a Collection variable could be:

> “Since the temperature is [[TEMP]] degrees .....”

The value of the variable [TEMP] will be embedded into the string when the string is added to the Collection variable.

Embedded variables can also be used in Prompt text to make it dynamically reflect the values in the system.

Whenever a string with embedded variables is used (string text displayed or used in an assignment or expression), and double square bracket embedded variable values will be immediately derived via backward chaining.

To embed a variable and have the immediate, current value be used without backward chaining, put an asterisk before the variable name in the square brackets. (e.g [[*variable_name]])

Once backward chaining starts, it will continue automatically. This may involve adding new variables as the top level goal as needed. It will continue until the Goal stack is empty.

Once a rule is in use as part of backward chaining, it will not be used to put the same variables on the Goal List more than once.

7.7 Controlling Backward Chaining

Once backward chaining starts by having a variable made the top level goal, the Inference Engine will normally fire every rule that can set a value for that variable. If there are multiple rules that can assign a value, they will **ALL** be tested and used if possible.

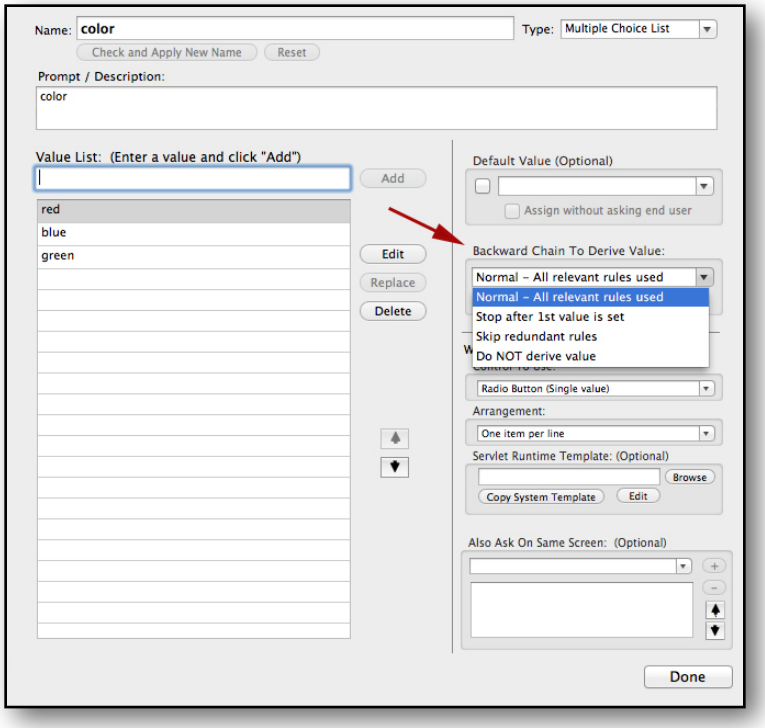
This is very convenient for probabilistic systems that have multiple rules that collectively set the overall value for Confidence variable.

Normally the default operation for backward chaining is exactly what is needed to run rules, but there are special cases where it is better to override how backward chaining works for specific variables. In some cases, using all the relevant rules can lead to unnecessary end user questions.

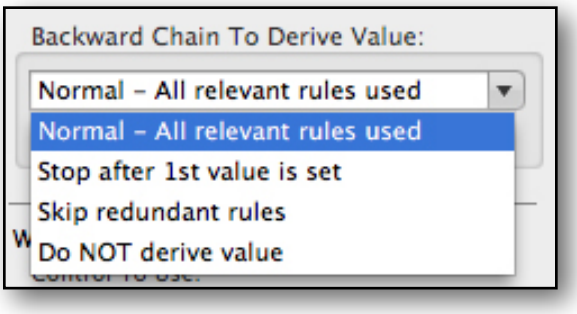
Sometimes there are multiple rules that can set the value for a variable but the rules are all logically equivalent - meaning that once one fires and sets a value, the others do not add any more information. In this case, once one of the rules fires, the other rules are unnecessary and any questions associated with those rules are unnecessary and should not be asked.

Individual variables can have their backward chaining options set to prevent asking the unnecessary questions.

When a new variable is added, or an existing variable edited, on the right side of the variable window, there is a drop-down for “Backward Chain to Derive Value”.



The default is “Normal - All relevant rules used” and that is **usually the correct selection.** But there are other options.



Stop After 1st value is set - Use backward chaining, but stop after the first rule fires that sets a value for the variable. This is useful if there are several independent and equivalent ways to derive the value and once one has been used, the others are redundant and not needed. If a rule is tested but is not True and does not fire, the system will continue until it finds the first rule that is True and fires.

Skip redundant rules - This applies only to List variables. A rule is “redundant” if it would not add any information to what is already known about a particular variable. It is similar to “Stop after 1st” but will fire additional rules that might set other values from the variable’s value list.

For example, suppose there is a List variable, [Color], with possible values “red”, “blue” and “green”.

If a rule fires in Backward Chaining that sets the value “Color = blue”. Any other rule that only assigned “Color = blue” in the THEN part would be redundant. It would not add any new information on [COLOR], so there is no reason to test or use it. Using the “Skip redundant” option would cause that rule to be skipped when the system already knows the color is “Blue”. On the other hand, a rule assigning “Color = red” in the THEN part would add information, not be redundant and would be tested. If that rule fired, the value of [COLOR] would be “red” and “blue”.

“Skip redundant” can be a good option to set for List variables with complex backward chaining derivation rules unless they are being used to control when a block of rules fire.

Do Not Derive - disables backward chaining for this variable. It is rarely needed, and is primarily for numeric variables used for counting that are incremented in many places in a system. For example, if there are several rules of the form:

IF

…

THEN

[COUNT] = [COUNT] + 1

backward chaining would create a large goal stack for [COUNT], and while that is legal and will work, is not needed.

Using the “Do NOT Derive” option would prevent the chaining and, provided [COUNT] was initialized to 0, would simply increment it for each rule that fires.

Remember, if the “Do NOT Derive” option is selected for a variable, it should not be used as a backward chaining goal. The command DERIVE [COUNT] would not cause rules that set a value for [COUNT] to be called. Instead, [COUNT] would receive its initial value, or if there is none, it would be asked of the user.

Running with Trace

Backward chaining does a lot invisibly. In most cases, the developer just adds the rules and the Inference Engine will take care of the rest. If you want to see the details of what and how it is running, you can always run with trace turned on and it will provide many more details. Trace displays the backward chaining goal stack, current active rule and status of all variables. Trace is covered in chapter 13

7.8 Which Approach to Use

Backward Chaining is a very powerful technique, but like any solution, it is not necessarily ideal for every problem. Some systems implement a process that is very procedural, and forward chaining may be a better solution.

One of the first steps in building any Corvid system is to decide if it will be primarily a Forward or Backward Chaining system or use a combination of both approaches. This decision controls much of the underlying architecture of the system and how rules and variables will be structured. Some simple systems can work well either way, but most complicated problems lend themselves more to one approach or the other.

Some factors to consider in making the decision:

1. Systems that interact with the end user in a dynamic way, skipping unnecessary questions but asking more detailed questions when appropriate typically use backward chaining to some degree. This may be in conjunction with forward chaining using the ALLOW_DERIVE option with a FORWARD command or as part of a pure backward chaining system run using a DERIVE command.

2. How is the knowledge that will be incorporated in the system currently documented? If there is an accepted and documented way to solve the problem, it is best to build the Corvid system in a way that follows that documentation as closely as possible. If the existing documentation is described as a series of steps, operations or tests that must be done in a particular order, it will probably be easier to implement this in Corvid using forward chaining since it makes it easy to describe the steps in a matching order.

   If some of the steps require asking the end user for input that should be derived from other facts, use the ALLOW_DERIVE option with the FORWARD command and build Logic Blocks that can derive the needed values. Use the FORWARD command to run only the Logic Blocks(s) that implement the procedural steps.

3. Is system implementing logic that is documented as a single tree diagram (or something equivalent to a single tree diagram) and the end user input will take it out on only a single branch. This can be done most easily with forward chaining, although backward chaining can also be used if all branches end with a variable (typically a Collection or Confidence variable) that can be used as the goal for backward chaining. While either can implement the documented logic, backward chaining makes it easier to add future enhancements to the system such as deriving value and implementing “I’m not sure” answers.

4. If the logic is implemented as multiple tree diagrams that are linked by having some branches end with “Go to ...” that require jumping to another tree diagram, use backward chaining. See Chapter 10 for how to implement this type of logic. Also use backward chaining for “logic” diagrams that loop. In a true tree diagram, each decision branch point should have only one input. If the diagram has multiple inputs to nodes, use backward chaining.

5. If the logic selects one item out of a group of possible items (diagnoses, actions to take, products, recommendations, etc) and there are multiple factors (rules) that contribute to deciding if each individual item is “best” or should be displayed to the end user, it is generally best to use

backward chaining. This is typically done with a set of Confidence variables that cover the possible items. Then use DERIVE CONF to have the Inference Engine use all the rules to set the value for all the Confidence variables and then the ones with highest values can be displayed in the results. This allows the various rules used to be arranged in any order and structured into various Logic Blocks.

6. If the system is building a report, use a Collection variable to build the content. This is easiest to do with a template that has embedded variables in double square brackets (see Sect 7.6). Double square bracket embedding automatically starts backward chaining, so reading in the template will automatically start backward chaining for any embedded variables. This will ask questions in the order that that data is needed - which may be OK. To change the order of questions that ALWAYS should be asked at the start, add ASK commands in the command block before reading the template into the Collection variable.

   If the report is being built by having rules add pieces of content to the report, and it is not built from a template (or large sections are not built from a template), and the content in the report must follow a particular order, it may be better to use forward chaining to add that control. Another option is to use a small template file to build the section, and then embed that into the larger report template.

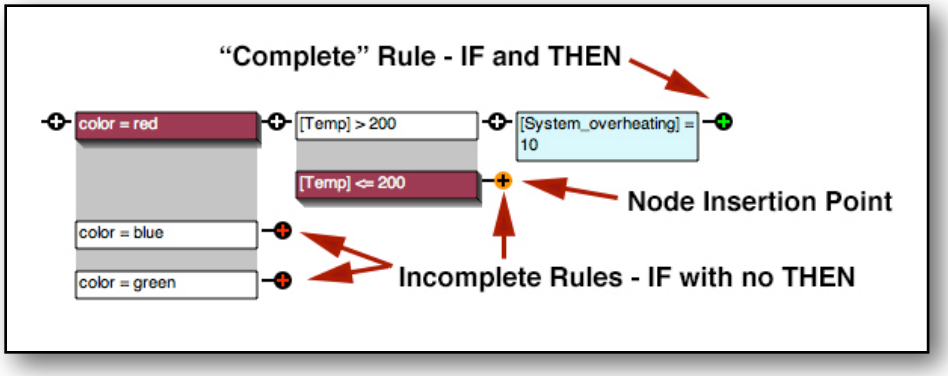
7. If there is no existing documentation to follow and the system is being built directly from the human expert explaining the how the decision is made, use backward chaining. This will make it much easier to implement more free form logic and fill in gaps. It also makes it easy to add rules to derive information needed in higher level rules.

8 Human Rules into Logic Blocks

Corvid allows describing the steps in a decision-making process in a way that is both readable by humans but still usable by the computer to drive an interactive on-line session. Humans are very good at understanding generalities and filling in gaps in logic in a reasonable way - computers are not. Computers need every step spelled-out in detail.

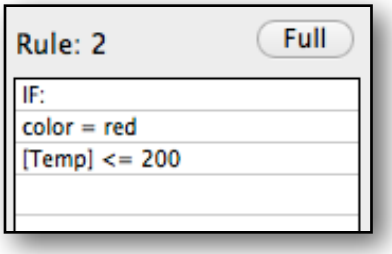
Corvid allows describing IF/THEN rules in English (or whatever language you prefer) and Algebra. Well written rules are easy to read and understand. The Inference Engine processes these rules to determine what is relevant to the current goal, what questions to ask and when an answer has been found. **However, the set of rules must be complete and cover all possible cases**. The Inference Engine can use the rules dynamically to reach a conclusion, but it cannot fill in gaps in the logic - this is still a computer.

Logic Blocks help to organize rules and make sure that they are complete. Backward chaining provides a way to write more general rules that call other rules to derive information, but both the high level rules and subset(s) used for backward chaining still need to be complete.



Building Logic Block trees from the top down and adding branches for all possible user input values, or all logically distinct cases, expands the tree with incomplete branches to remind the developer that those situations need to be covered in the logic. Logic Block branches end with a red or green node. Red means that the branch has IF node, but no THEN nodes and is incomplete. Once a THEN node is added, the ending node becomes green. This means the branch is "complete" in that it has both IF and THEN nodes, but your logic may still call for additional THEN nodes to be added.

In addition, the active insertion point will be highlighted in orange. This is automatically moved to the end node on the current active branch as nodes are added. The orange of the insertion node can hide the red or green end node, but looking at the Rule View window will always show the rule up to the insertion point. Also, THEN nodes are always blue, so each branch should end in at least one blue node.



It is actually not illegal in Corvid for a branch to not have any THEN nodes, but that branch will not build a rule and not have any effect on the logic or running of the system. However, there is no problem running and testing systems that do not have all branches complete, just remember that there is no logic or rules for some paths in the system. Some more advanced systems can use "incomplete" logic to trigger actions for user input that indicates a problem outside the scope of a system. This can trigger actions such as elevating the problem from the expert system to higher level human support staff.

However, most systems are designed to have Logic Block nodes that completely cover all reasonable user input and block "unreasonable" input either by restricting the options for list variables or by applying limiting numeric, string and date values.

8.1 Use Clear Questions

Logic Blocks should be designed with variables that will ask end users questions that they can unambiguously answer, and should handle each of their possible answers in the logic. Questions should be clear, precise, easy for the end user to answer and relevant to the intent of the system. For example, asking the end user if something tastes “good” or “bad” would generally be a poor question since it is so subjective. However, for a system that was actually looking at that user’s subjective opinion of known items, it would be OK. Asking for a temperature as a numeric value should only be done when it is reasonable to expect that the user will have instrumentation available to measure the temperature. Otherwise it might be better ask if the item is “warm to the touch”, “too hot to touch”, etc,- while somewhat subjective, may provide enough differentiation for the purposes of the logic.

If the user’s input may be in various possible units (e.g. temperature in centigrade or fahrenheit) The question should specify the units or give the user options that they can select and convert internally when needed.

When adding the prompt for a variable, always think - “Can my intended end user answer this easily and unambiguously?” If the intended end user is knowledgable about a subject, more complex questions can be asked. For example, a system intended for doctors can use medical terminology that would not be appropriate for the general public. If the end user may not know how to answer a List variable question, include an “I’m not sure” answer and use backward chaining to derive the answer (This is covered in detail in section 10.3)

High level logic in a system often is written in terms of variables that are not appropriate to ask the end user - but in these cases, backward chaining should be used to ask more appropriate questions that the user can answer, and which will be used to derive and set the value for the higher level variables.

Designing good questions (variable prompts) is key to building systems that end users can work with. Test systems with typical users to see if they understand and can answer the questions.

8.2 Cover All the Bases

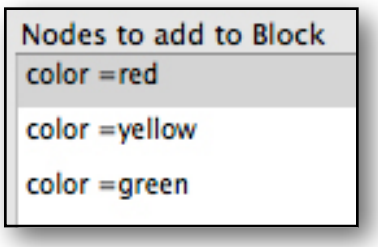
The logic in the system should be designed to handle any value that the end user inputs. This is fairly easy with List variables since Corvid will build a branch in the Logic Block tree for each value or group of values. When building a node for the rule currently being worked on, add the other possible values to create branches for those values. Having the branches in the Logic Block reminds you to fill them in later, even if you do not have the specific logic for them at the moment.

Generally, when building a node with a list variable, each of the values will get added to the “Nodes to add to Block” window either individually or as part of a group of values that are logically equivalent.

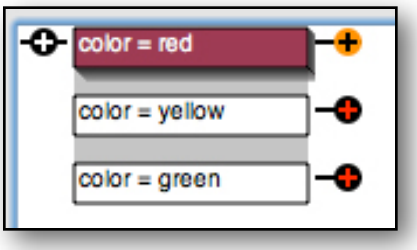
For example, for a List variable [color] with values of:

- red
- yellow
- green

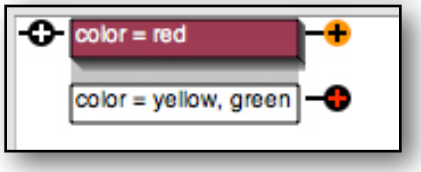
If a node was being added to a Logic Block for the “red” value, nodes would also be added for the “yellow” and “green” values. This is done by adding each to the “Nodes to add to Block” either one at at time, or with the “Each” button.



This will build one node for each value in the tree which can be expanded into individual rules.



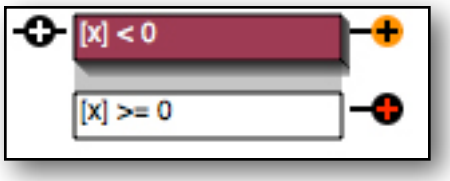
When multiple values have the same logical meaning in the system, and would be used to build identical rules, they should be combined in the "Nodes to add to Block" window by selecting them all and using the "Selected" button, or if they are the only values remaining, using the "All" button. For example, if the yellow and green values have the same meaning in the logic, they should be combined in the same branch to avoid redundant rules (which while legal, is not a good design practice).



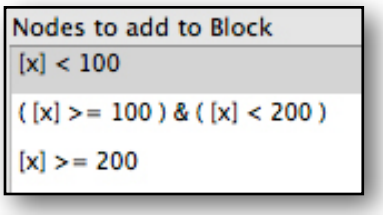
Also, It is not required to include each List variable value in a branch. If you know that a particular value has no logical implication, it can be ignored and not included in the tree at all. It is legal to include it and not give it any THEN nodes so it will be ignored by the Inference Engine, but since that will appear as a gap in the system logic, it can be confusing.

For example, suppose a diagnostic system knows there is a special situation for the "Model 123" machine. It could have a List variable [Model_123], that asks if the machine is the Model 123, with values of "Yes" and "No". If it is, there are other rules that need to be considered, but if not, it does not add anything. In a case like this, just build the branch for the "Yes" node and ignore the "No" node.

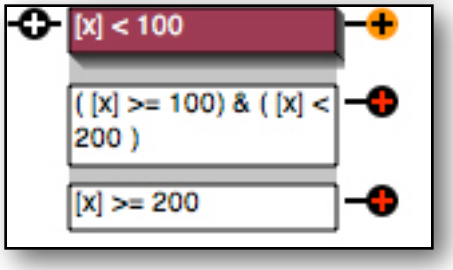
Building complete sets of nodes for non-List variables is done the same way, but requires building a set of expressions to cover all the possible cases. For simple expressions this is easy. For example, suppose the system needs to check to see if the numeric variable [x] is less than 0. Just build nodes for "[x] < 0" and the node to cover the other possible values "[x] >= 0". This will build 2 nodes in the tree and all possible values of [x] are covered by one of the 2 branches. (Note: Remember to add the >= rather than just > to cover the situation where [x] is 0).



To check if [x] is in the range of less than 100, 100 up to 200 or greater than or equal to 200,. build 3 nodes using parenthesis to make the expression clear and & (logical AND) to combine multiple tests.



This will build 3 nodes in the tree to cover all values.



For simple expressions with a single test, this is fairly easy, but for complex expressions with logical OR like:

(([X] > 9) | (([Y] < [X]) & ([Z] < ([X] / [Y]))))

It can be very difficult to design a set of expressions to cover various possible values. Fortunately, for expressions like this it is usually a test of the expression being true or it not being true. This can be achieved by simply using logical NOT - the ! character, in front of the expression to create a pair of expressions that will cover all values:

(([X] > 9) | (([Y] < [X]) & ([Z] < ([X] / [Y]))))

! (([X] > 9) | (([Y] < [X]) & ([Z] < ([X] / [Y]))))

8.3 Limit the User Input Values

In some cases, the meaning of the variable limits the possible values that make sense for the end user to input. For example, suppose the variable [Count] is the number of items in a box. The value must be zero or greater and must be an integer. In this case, the nodes can be written to only cover the relevant cases such as:

[Count] = 0

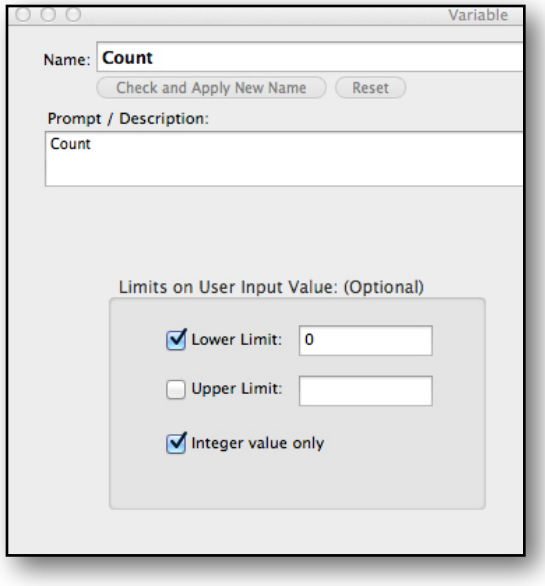
([Count] >= 1) & ([Count] < 10])

[Count] >= 10

These do not cover values of [Count] that are less than 0 or between 0 and 1. In the real world, this is not a problem since those would not occur, but remember that end users may put in values that are not realistic to see what the system does. This can be prevented by adding logic to catch these unrealistic values, but that complicates the logic. It is better to simply block the values from being input. This is done when setting the properties for the variable.

Since [Count] is numeric, it can have optional upper and lower limits and an option to only accept integers. If the end user running the system, inputs a value that does not match the input limits, their input will be rejected and the question re-asked. In this case, we could set the lower limit to reject any negative value and limit the input to only integers. An upper limit could also be set for the maximum number of items that could fit in the box.

String and date variables also have limits that can be set to restrict the end user's input. When appropriate, these can greatly simplify the number of nodes needed to still handle all input values.



8.4 Building Complete Logic Blocks

There are 2 main approaches to building a Corvid expert system. One is a system that implements existing written documentation on how to solve a problem, the other is building a system from more "free form" knowledge that is only in the head of the human expert.

The first is much easier since there is only the single task of converting the written documentation to Corvid form.

Tree Diagrams

The easiest case is when the existing documentation is already in the form of a tree diagram. This can be a decision tree, dichotomous tree, or any other tree structured representation of the decision points and steps that are needed. In this case:

1. Build a variable for each decision branch point in the tree.
2. Build out a Logic Block with these variables matching the structure of the existing tree
3. If there are "informational" or instruction steps in the **middle** of the logic that just require the end user to do something without it being decision point, build a List variable where the prompt tells the end user what to do, and give it a single value of "OK". Add these to the tree at the appropriate points **as IF nodes with the single OK value**. They are not decisions, but will still fit in the tree and require the user to click "OK" to proceed out of the tree. If there are action/ recommendations at the end of a branch, make them THEN nodes.
4. If there are many different end nodes (actions / recommendations) in the tree, create a Collection variable and just add the advice/recommendation into it as text at the end of each branch. This will provide the advice that can be displayed to the end user when that branch (rule) fires.

If the "tree" is actually multiple tree diagrams linked with "Goto" nodes the same approach can be used, but the trees need to be linked. See Chapter 10 for the details on linking this type of trees.

Regulations and SOP

Regulations and operating procedures are often written in a form that is equivalent to IF/THEN rules, even though it may be phrased more as "When... do.." or something similar. These can be converted to Corvid form, but it is not quite a simple as tree diagrams.

1. Convert each specific regulation that describes a decision point into a variable(s) that implement that decision in an IF/THEN form. Add these to a Corvid Logic Block. Use each of the possible values for the variable or expression to build out the additional related branches in the tree. With each existing regulation, see if its IF conditions match one of the incomplete branches to complete it and fill out the tree structure.
2. As with trees, convert any step in the middle of a branch that is informational or just calls for an action with no real decision, into a List variable with just an OK value.
3. Related regulations can be structured in a tree if they are based on different values of the same variables. However, often there are completely independent sections of regulations that need to be built in different trees. Just handle each in its own Logic Block.
4. Build out the tree(s) . The end of a branch may be a simple "Your Done" message or may have specific actions to take. If various parts of the regulations are independent and the user may have multiple branches (rules) fire, use a Collection variable and have each branch add to its value which will be displayed to the end user. **Make sure that each part of the regulations has a corresponding branch (rule).**

One of the biggest problems with converting regulations to Corvid rules is that the Corvid tree structure can disclose gaps in the logic of the regulations. This is because regulations may be designed to cover certain scenarios, but have gaps as to what should be done in other slightly different cases. For example, a regulation may say “IF A and B and C, THEN do X”, but what about when there is “A and B, but not C” or “B and C, but not A”. Often regulations are built by committee and sometimes it shows.

If there are branches in the tree(s) that end with a red “IF with no THEN” marker, there will be combinations of input that lead to no conclusion, and the core knowledge is not in the existing regulations/ SOP. Generally this means an authoritative human expert needs to be consulted to fill in the gaps. While this knowledge can be difficult to obtain, it can result in a Corvid system that is both more complete and definitive than the original regulations.

8.4 Knowledge is Only in the Expert’s Head

A major problem throughout business and industry is the loss of expert knowledge. While certainly not everything a human expert knows can be converted into an expert system, many decision making tasks that are well understood and “routine” to the human expert may be suitable. Even though these tasks may seem routine to the expert, non-experts may have a very difficult time with them, and capturing the experts knowledge can have tremendous value.

The most unstructured (and sometimes difficult) type of Corvid system to build is when there is no written documentation of the decision making process - however, this is also the most valuable type of system to build since it can document and preserve valuable expert knowledge.

The best approach in this case is when the expert is using Corvid to build their own system since they can most easily build and test rules directly. Corvid is designed for non-programmers and is easy to learn. Often it is far quicker to have the human expert to learn Corvid and build, at least, the core decision-making logic. However, it can also be done working with a “knowledge engineer” that knows Corvid and can use it as a way to structure the expert’s knowledge.

Usually the best approach to a problem like this is to think of how would the expert teach an apprentice (or new hire) how to make the decision. At the highest level, what are the IF/THEN rules that the expert uses to make a decision. These can be built out in Corvid. Build them using List variables with the various possible values, or groups of expressions that cover all cases, to make the tree expand to cover all cases. This can be used as a way to lead the expert to consider what should be added on the various branches - do they lead to a conclusion or other questions that would need to be considered. If the questions in the top level tree(s) are not ones that could be asked of end users, build lower level rules derive the values of the high level variables using variables (questions) that the end user can answer, and let backward chaining link in the other trees as needed. In some cases this may need to be done multiple levels.

Another approach is to look at specific decisions the expert made and ask “Why did you make that decision”. Again start building out Logic Blocks based on the reason for the steps taken, with additional branches added for all the possible List variable values or expressions. This can then be used to ask the expert about what would be done on the other branches.

Once a reasonably complete set of logic is built, test it with the expert for various cases. It is likely they will say “that’s not right, it needs to also consider ...”. Testing real cases with the expert is the best way to find sections that need to be expanded. Then go back into the logic and make whatever changes are needed. If any branches end in a red “IF with no THEN” marker, they need to be considered and, where appropriate, completed.

Because this is a rather unstructured way to build a system, it is likely the first attempt, while providing valuable knowledge, may prove not the best way to approach the problem and need to be restructured. Often if a system is getting too convoluted, it is best to start over and rebuild it based on what has been learned. In this case, the variables that were created can generally be reused and only the Logic Blocks need to be deleted and rebuilt in a different structure.

9 Working with Confidence Variables

A particularly valuable type of Corvid system is one that captures probabilistic knowledge used to make a decision. This is one where the decision is based on balancing multiple, sometimes competing, factors to arrive at the overall “best” recommendation(s) or most likely action(s) to take. There may be no exact answer, and the basis for the decision complicated by combining various rules.

This type of decision making knowledge is rarely well documented simply because traditional approaches do not work well for documenting problems where many separate factors are involved and need to be considered and balanced. Usually this knowledge only comes from having made the decision many many times for various cases, with opinion on what worked well and what did not - real expert knowledge of a complex task. Corvid provides a way to structure and capture even this highly valuable knowledge in a way that allows it to be put online.

The key to building probabilistic systems is Corvid’s Confidence variables. These are special Corvid variables that allow multiple rules to each contribute different “confidence values” that are automatically combined by Corvid to an overall value for the Confidence variable that measures how “good” or “likely” or “suitable” the advice/action represented by that variable is. There is normally a set of Confidence variables for the various recommendations that the system could provide and the one(s) with the highest overall value are the ones that are displayed to the end user as the “best” action to take.

Systems using a set of Confidence variables work extremely well when there is logic that calls for many independent trees for various factors in a decision that all must be combined. They work well for probabilistic diagnosis where there may be several possible causes for a problem based on characteristics/logic that each suggest, or eliminate, a particular cause. They also work well for recommending the “best” option from a set of items to meet user criteria, where no item may be a perfect match.

Probabilistic logic can be difficult to understand at first since it is not found in most other programs. First, remember that it is a “Confidence” variable with a “confidence value”, not generally a true probability in the statistical sense (though it can be when appropriate). This is because most processes do not have detailed statistical data on events and outcomes that can be used with the formal mathematics of probability, and applying rigorous probability formulas would not be valid. Most human expert knowledge is based more on overall experience and not on formal statistics. An expert may know that “seeing X makes it very likely the problem is Y”, of “if X is happening, Z is almost never the cause”. The goal with Corvid Confidence variables provide a way to convert the “very likely” and “almost never” into a numeric “confidence value” that represents the expert knowledge and can be used by the Corvid Inference Engine to reach valid overall conclusions.

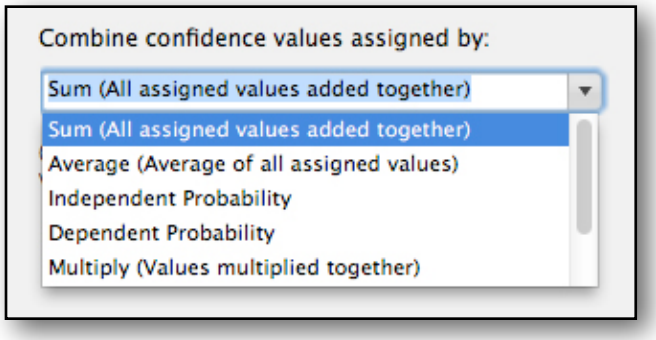
9.1 Combining Confidence Values

In most respects Corvid Confidence variables look and behave like numeric variables. They can be used in formulas, expressions and assignments anywhere a numeric variable could. For example, if [Conf] is a Confidence variable and [X] is numeric, the following could be used:

- [Conf] = 5
- [Conf] = [X] / 3
- [X] = [Conf] * 2
- IF: [Conf] > 4

The BIG difference between Confidence variables and numeric variables is that when a numeric variable is assigned a new value, that becomes the value and any previous value is completely forgotten. In contrast, Confidence variables remember **all** the values that they are assigned and combine them to a single overall numeric value based on the Confidence mode selected.

When a new Confidence variable is added to a system, the mode (algorithm) used to combine values must be selected. Corvid provides 7 ways to combine the individual values. Each variable can have its own way of handling confidence allowing multiple techniques to be combined as needed. However, most systems using confidence will have a set of Confidence variables that all use the **same** mode to combine confidence values.



A mode should be selected that matches the approach used by the human expert providing the decision-making knowledge for the system.

The mode for combining values is selected from the drop down list. There are 7 options:

Sum - The values are added together. Positive values increase the overall confidence, negative values decrease the overall confidence. This is a simple system, but works very well for many systems. Unless there is valid statistical data, this is often the best way to combine "rule of thumb" factors in a decision.

Average - The values are added together as with "Sum", and then divided by the number of values. This provides another simple way to combine competing factors, with individual factors having less influence when there are many values added.

Independent probability - The values are combined as if they were independent probabilities. If there are values X and Y, the combined value will be 1 - ((1-X) * (1-Y)) The individual values must be between 0 and 1. This is a statistically more rigorous approach, but requires that there be valid statistical data that can be applied.

Dependent probability - The values are combined as if they were dependent probabilities. If there are values X and Y, the combined value will be X * Y. As with the Independent mode, values must be between 0 and 1, and it requires having valid statistical data.

Multiply - The values are multiplied. This is similar to the dependent probability mode, but here there is no assumption that the values actually represent probabilities, and the values can be any positive value in any range. For example, a rule could lead to doubling the confidence by giving it a value of 2, or halving the value by giving it a value of .5. Values assigned in this system should be positive.

Maximum - returns the largest value assigned. This is useful for cases where individual rules can flag a variable as "good", regardless of lower values from other rules.

Minimum - returns the smallest value assigned. This is the opposite of Maximum. It is good when a rule can eliminate a variable by giving it a low value, regardless of high values given by other rules.

The values assigned to a Confidence variable can be a simple numeric value or come from an expression that evaluates to a numeric value. The expression can even include other Confidence variables, so confidence values calculated in one part of a system can be "propagated" to influence the value of other Confidence variables. Using backward chaining, the confidence values needed to make an assignment will be fully derived before making the assignment, so rule order is not generally important.

Regardless of how the value assigned is defined (simple value or expression), ALL of the values assigned to a Confidence variable will be combined based on the confidence mode for that variable, and **ALL** rules that can contribute to setting the value for a Confidence variables (value assigned in the THEN part of the rule) will be tested and used to set the Confidence variable via backward chaining.

So if a Confidence variable is assigned the values -3, 4 and 7 in various rules its overall value will depend on the confidence mode selected for that variable:

Sum	8
Average	2.6666
Maximum	7
Minimum	-3

The independent and dependent probability modes could not be used since the values are grater than 1 and Multiply could not be used since one value is negative.

The Sum system, while simple, is adequate for most confidence systems and the “add or subtract points” approach is easy to work with and understand. If there is statistical data on a process, the Independent or dependent modes may work. If there are primarily thresholds for including items, the Maximum and Minimum approaches can work well. The goal is to use an approach that matches the way the human expert thinks about combining and weighting the various possible outcomes or items.

Usually, unless the way the problem is currently solved calls for one of the more rigorous or specialized modes, the Sum mode is a good place to start. If the logic gets complicated or this mode does not provide the needed differentiation between variables, switch to another mode.

There are many ways Confidence variables can be used. Some systems use them just to organize the possible recommendations that a system may display, even though they are not really confidence values and only a single one will be selected. In this case, just set a “flag” value to the appropriate Confidence variable at the end of a branch. Using the “Sum” confidence mode, this can be any positive value. Then just select to display the Confidence variable(s) with a value greater than 0.

For example, a regulatory system might select the forms that need to be completed for a transaction. There could be Confidence variables “Form A”, “Form B”, ... and some of the rules might be:

```
IF
        [transaction_amount] > 10000
THEN
        [Form_A] = 1
and:    [Form_B] = 1


IF
        Buyer_is_outside_of_US = YES
THEN
        [Form_B] = 1
```

Then just display the Confidence variable that get a value greater than 0. If both rules fire, [Form_A] will have a value of 1 and [Form_B] will have a value of 2. It does not matter that Form B is required based on 2 criteria, it just needs to be completed.

These 2 simple rules could be folded together to a more complex set of branches, but when there are many rules and criteria that need to be considered, it can get very complicated. Also, but having a single rule that matches each criteria is easier to build and verify.

The more powerful use of Confidence variables is in situations where the confidence values actually have meaning and reflect the likelihood of various items applying. For example, a diagnostic system might have rules:

IF

Computer_is_making_a_rattling_noise = YES

THEN

[Problem_is_the_fan_mount] = 10

and: [Problem_is_the_fan_bearing] = 5

IF

Overheat_light = ON

THEN

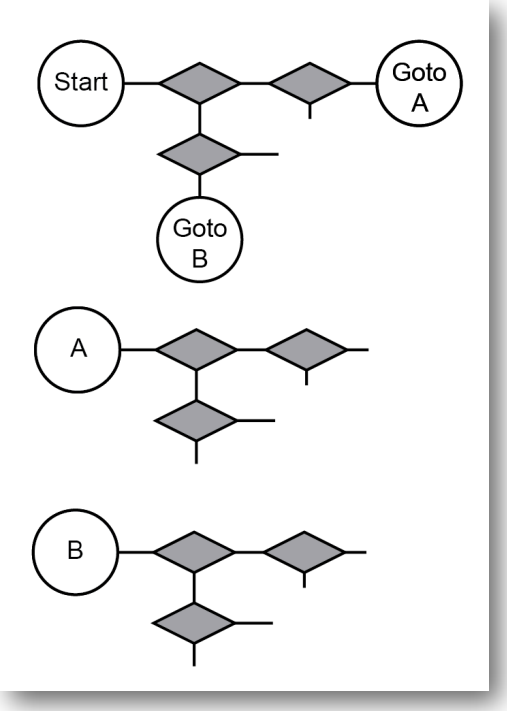
[Problem_is_the_fan_bearing] = 10

Here the first rule says that a rattling noise is most likely due to the fan mount, but could be the fan bearing. The second says that if the overheat light is on, the system will add “points” to the value for the bearing. Combining these 2 rules with the Sum confidence mode will assign the Confidence variables values to indicate the most likely cause of the problem.

10 Linked Tree Diagrams

10.1 Handling Linked Tree Diagrams

A standard way to document complex decisions is with tree diagrams. Since there are limits on how large a tree can be printed on a page, a common technique is have the end of a branch be a “Goto” that indicates that the user should start at the beginning of another tree.



For example, here are 3 trees. The top one has branches that end with “Goto” nodes that indicate which of the other trees the user should then start at.

Even though they are linked, these are fundamentally tree diagrams and fit naturally with Corvid’s Logic Block structure. However, the links require a few additional steps.

One approach would be to build a big tree in Corvid. Since Logic Blocks have no size limitations, the “A” and “B” tree nodes could just be pasted into the main tree at the Goto points to build out a single large tree with no “Goto”. If the final tree is not very big, this can be a good way to approach the problem - especially if the initial tree was only broken up to make it fit properly on a printed page. Hover, often the resulting tree would get so large that, while it would work, it would be inconvenient to navigate, edit or maintain.

Remember, it is usually best to have the Logic Blocks match the existing documentation. It would be more difficult to make sure the larger merged Corvid tree has all the branches filled in correctly. Also, if there are multiple “Goto” nodes that select the same tree, (e.g. Several different paths where the user should skip to tree “B”), some parts of the tree will have to be repeated. This is generally bad design if it can be avoided, and any future change in the repeated tree would have to made in several places.

Corvid has a way to make this type of link tree structure easy to implement. All that is needed is to add a List variable for each tree, except the first, that controls if it should be used. Once this is done, build each tree in a separate Logic Block. The set of trees can be run in either backward or forward chaining (which would be determined by other factors). However, if run in forward chaining, the trees must be in order in their associated Logic Blocks and “Goto” links cannot go back to a preceding tree. If run with backward chaining, the individual tree Logic Blocks can be in any order and can link to preceding or following trees. (If the logic structure requires going back to previous trees, be sure to run with backward chaining) Also, individual trees function as independent modules and can be called from as many places as needed. Changes in an of the individual trees will automatically carry though whenever that tree is called. This approach can be nested as many levels deep as needed.

For this explanation, the first tree is called the “Main Tree” and the others are “subtrees”, however this approach also works when there are more than 2 layers and the “subtrees” also end in “Goto” to other subtrees.

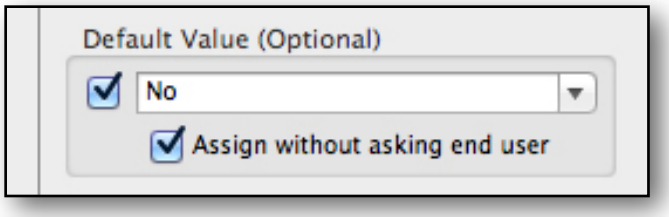
1. Add a List variable for each of subtrees. This should have a name something like [Run_Tree_A], [Run_Tree_B] etc. This will control which trees are used.

2. Each of the [Run_Tree_X] variables should have values of “Yes” and “No”.

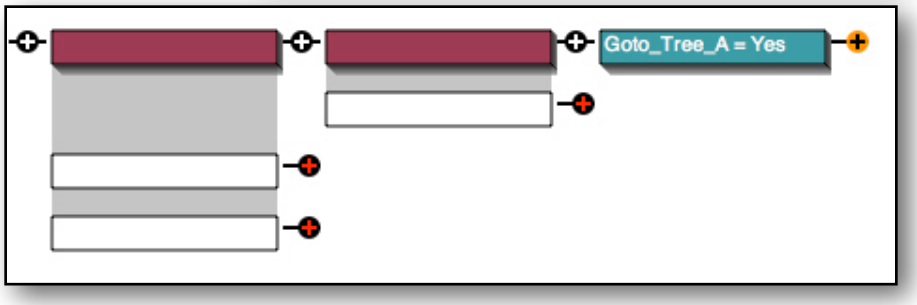
3. **This is the key.** When adding the variables, in the upper right corner of the window, select the Default value. Set the Default value to “No” and check the “Assign without asking end user” checkbox.

   Setting the default value means that when the variable is asked, it would have this value preselected. However, adding the “Assign without asking end user” option means that when the variable is needed, the selected Default value will be immediately automatically assigned without asking the end user to input a value.

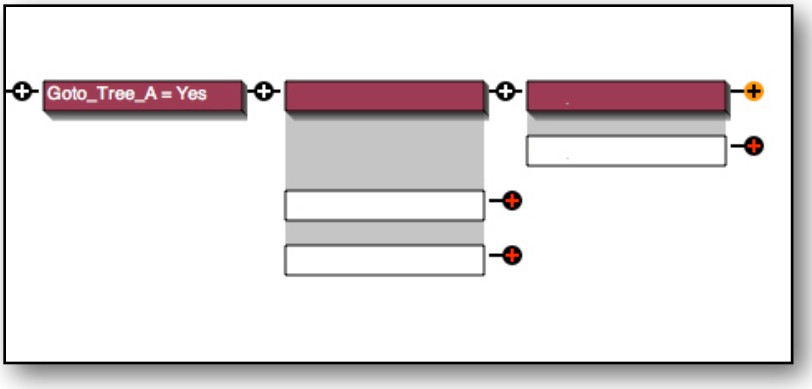
   Unless a rule specifically sets the value to “Yes” the default “No” will keep unselected trees from running.



4. Create a Logic Block named “Main” and build out the main tree logic as nodes, adding variables as needed. When the “Goto X” node is reached, use the variable [Run_Tree_X] and set the value to “Yes” in a THEN node.



5. Create a new Logic Block for each of the subtrees. These can be named “Tree A”, or whatever matches the existing documentation. Make the first node in each of the Logic Blocks an IF node with “[Run_Tree_X] = Yes” for the associated variable.



   Do not add a node for the “[Run_Tree_X]=No” value - that value has no meaning. To the right of this node, build out the logic for the associated subtree adding variables as needed. The branches can end in “Goto” nodes to other trees if needed.

The trees can be run in Forward chaining if the tree order does not require looping back to a previous tree. Usually it is better to use backward chaining to DERIVE a Confidence variable (or other variable) that has its values set in the main or subtrees.

How It Works:

In Forward chaining, the Main tree is tested first. The user input may results in a rule firing that sets a "[Run_Tree_X]=Yes" (Equivalent to a "Goto X" in the original documentation). When the subtrees are tested, their first node is the IF test "[Run_Tree_X]=Yes". If the Main tree set this, that subtree is used. If the Main tree did **not** set that [Run_Tree_X] variable to "Yes", it is automatically set to its default value of "No" and that tree is blocked since all rules in the tree have the top IF condition "[Run_Tree_X]=Yes".

The end user will NEVER be directly asked for any of the [Run_Tree_X] variables. (If they are, the Default value and "Assign without asking end user" are not set correctly for that variable.)

In Backward chaining, the DERIVE command is used to tell the Inference Engine to get the value for a variable (typically a Confidence variable, although any type can be used). The Inference Engine will look for any rule that sets a value to the variable in the THEN part of a rule. This should be in the subtrees.

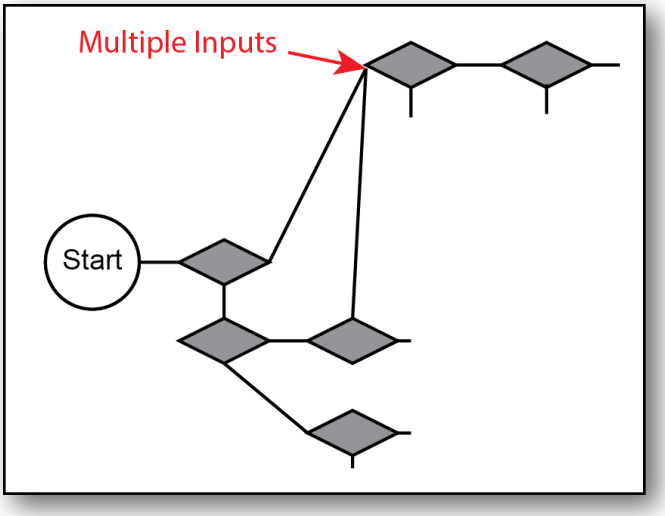
The rules found will have the first IF condition be the "[Run_Tree_X]=Yes" for the associated tree since it is the top node in the tree. Since this is running in backward chaining, this will cause the Inference Engine to look for any rule that sets [Run_Tree_X] in the THEN part of a rule. This will be found in the Main tree (although it can be in any tree), so the associated Main tree branch is tested. If it is found to be True and "[Run_Tree_X]=Yes" is set, the subtree can fire and set the value to the variable that started the backward chaining. If [Run_Tree_X] is not set in the Main tree, the default value of "No" will be assigned and the subtree will be blocked. This repeats for any other occurrences of the starting backward chaining variable(s).

The backward chaining approach may seem more complicated, but the complexity is handled by the Inference Engine invisibly and does not require any action by the end user. It allows trees to link together in any combination and at any level.

10.2 "Tree" Diagrams with Multiple Entry Paths

A true tree diagram will have only one entry path for any node. It may have any number of paths that branch out from it, but only one that goes in. Various drawing and diagram tools make it easy to draw "trees" with multiple entry paths, and many types of decisions are documented with these type of diagrams. These are not true "trees", but are still easy to build in Corvid.

When there are multiple points in a decision that can lead to the same actions or logic, it makes sense to reuse the logic that is already diagrammed. In linked trees, this results in multiple "Goto X" nodes where needed. However, smaller trees often represent this by having a single node with multiple entry points.

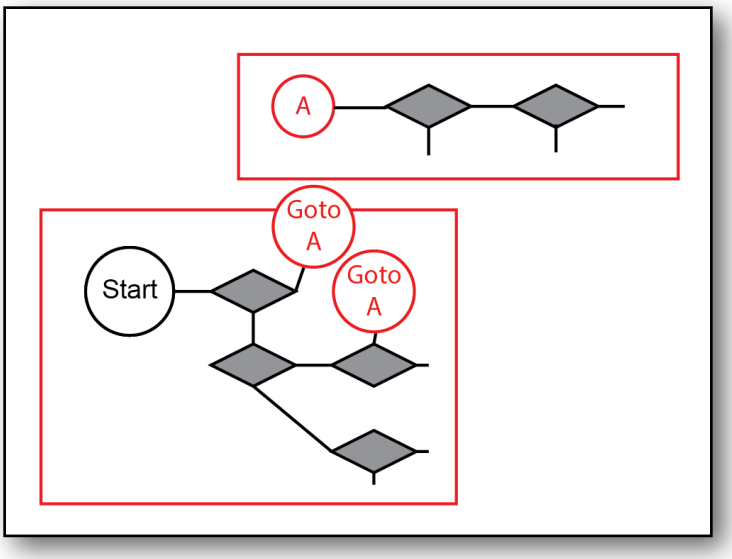


These can be handled in Corvid the same way linked trees are. It is just a matter of converting the multiple entry paths into linked trees.

Wherever a node has multiple entry paths, make it a starting point for a new tree (Logic Block)

Then make each of the entry paths be a "Goto X" end of the Main tree.

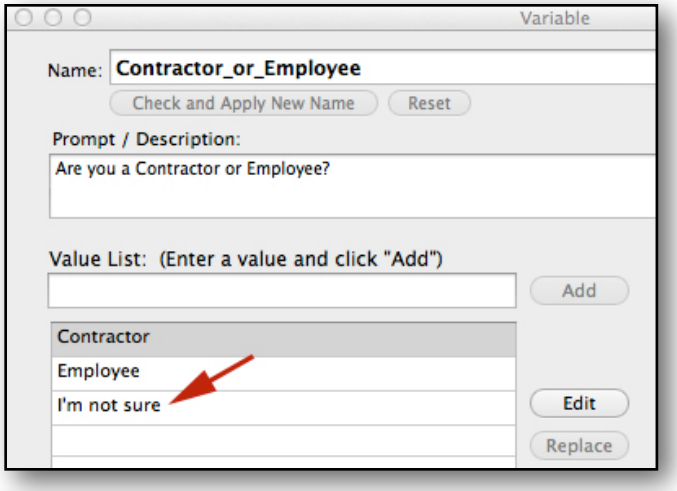
This splits the original tree into multiple linked trees that can be handled by the same technique described above for linked trees.



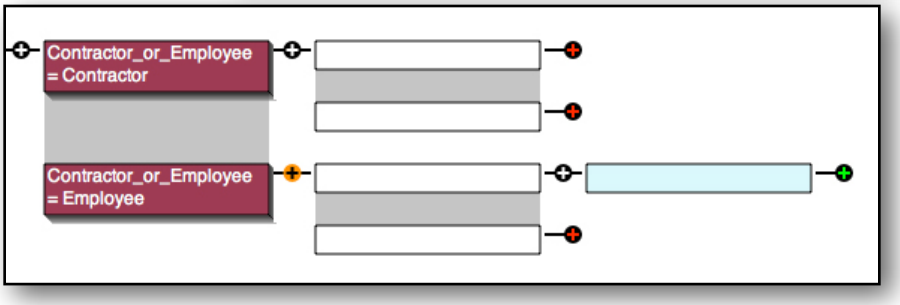
10.3 Implementing an "I'm Not Sure" Option

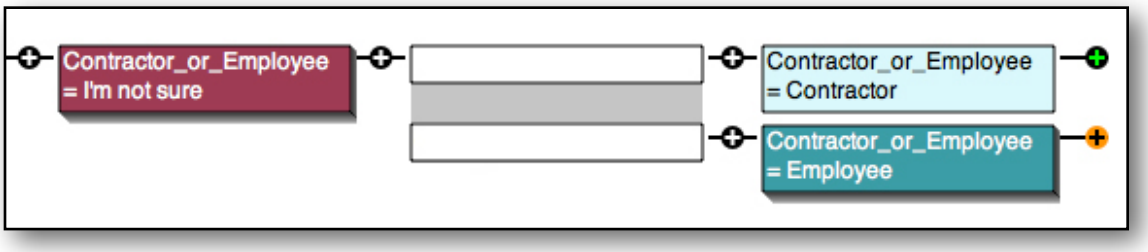
Backward chaining makes it very easy to build systems that allow the end user to answer "I'm not sure" to a question and have the system assist them. For example, a tax system might ask the user if they are a "contractor or employee". Many end users will know the answer and can answer it, but this can be a gray area and some end users may not be sure. Since this can be an important factor, you don't want the end user to be forced to just guess when they are not sure. Adding a "I'm not sure" option can solve this.

1. Create a List variable (For this example, [Contractor_or_employee]) and give it a list of associated values, plus an "I'm not sure" value



2. Create whatever high level logic as needed to cover all the values **EXCEPT** the "I'm not sure" value.





3. Start a new Logic Block and have the first top IF node check the variable for the “I’m not sure” value. This means the rules in that block can ONLY fire if the user answers “I’m not sure.” Add rules in the Logic Block that determines which of the other values apply based on lower level rules and criteria. This can be complicated logic, but the rules should each end with a THEN condition that sets the [Contractor_or_employee] variable to one of its other values. No branch should set the “I’m not sure” value.

4. Backward chaining will do the rest. When the value of the variable is needed in the high level logic, the rules from the new Logic Block will be automatically tested since they can set the value for the variable. Since the top IF condition in each rule from that block starts with “variable = I’m not sure”, the variable will be asked of the end user. If they answer one of the other values (NOT “I’m not sure”), all the rules from the new Logic Block are false and will be blocked. If they answer, “I’m not sure”, the new Logic Block rules will derive the value of the variable. It will end up with both a value of “I’m not sure” and one (or more) of the other values. Having the multiple values set is not a problem since this will still allow the rules in the other higher level Logic Blocks to work correctly.

Adding an “I’m not sure” value can make a system apply to a much wider range of users. Less experienced users can get the help they need by answering lower level questions, but expert users can simply answer the high level question without having to go through details they do not require.

The questions in any good system should always be something the intended end user can answer. Using “I’m not sure” can help to achieve this.

11 Command Blocks

11.1 Command Block vs Logic Blocks

Command Blocks were mentioned in the discussion of forward and backward chaining since command block commands are used to start rule execution. This chapter provide much more detail on the various command options that can be used to run a system.

Logic Blocks define the rules in a system. Those rules describe the logical relationships and heuristics that the Inference Engine can use. However, the rules by themselves do not do anything. It is the command block that tells the Inference Engine what to do and how to use the rules.

Logic blocks tell the system how to do things.

The Command block tells it what to do.

All systems must have a Command block. The Command block is a list of commands that are executed in order. Those commands may do something directly or may tell the Inference Engine to do something using the Logic Blocks. There are IF and WHILE commands that allow conditional execution of sections of the command block or looping.

An important thing to remember about Command Blocks is that they should be limited to procedural operations rather than logical ones. Logic should be put in the Logic Blocks - not the Command Block. Even though Command Blocks support IF tests, they should be used IF tests, they should be used for controlling flow for* procedural *reasons, not as an alternative way to write core system code.

Separating procedural flow from core decision-making logic makes a system much easier to build, understand and maintain. Also, backward chaining only looks at Logic Blocks for rules. It does not consider any logic that is in the command block. All decision-making rules should be kept in the Logic Blocks so the Inference Engine can find and use them.

Also, command blocks should usually be short - generally less than a dozen commands and often only 2 or 3 commands. Long command blocks typically indicate that they contain logic that should be in the Logic Blocks, and the system should be restructured to put the logic in Logic Blocks.

It is easy, especially for people with a programing background, to look at the command block as a scripting language and try to write the system in that. This is particularly common when first working with backward chaining, and not being accustomed to the Inference Engine automatically doing things that would otherwise need to be explicitly coded. Command blocks let you control and override what Inference Engine does. Sometimes that is necessary, but in most cases it is best to understand how the Inference Engine works and use it effectively.

11.2 Default Command Block

When a new system is started in Corvid it is given a default command block that can run the rules, but this will usually be modified later to a command block tailored to the specific system.

The default Command Block commands are:

```
FORWARD ALL ALLOW_DERIVE
RESULTS
```

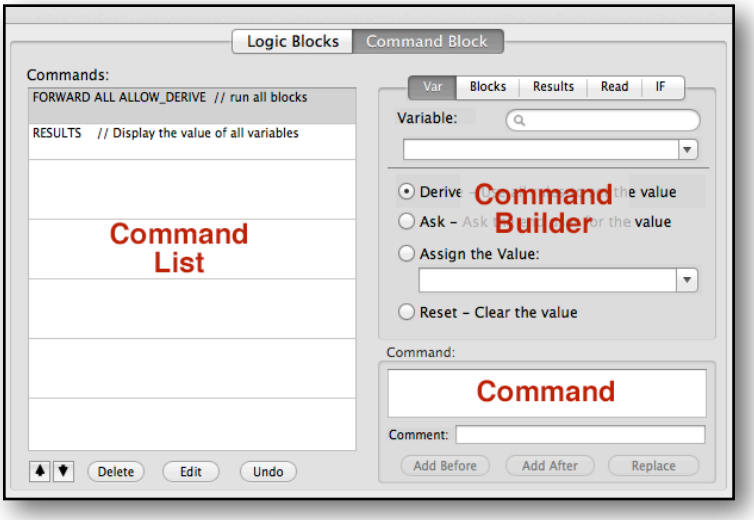
The FORWARD ALL command runs all the Logic Blocks in a system in forward chaining. Each rule in each Logic Block will be tested in order. The IF conditions will be evaluated and if all the IF conditions in a rule are true, the THEN assignments associated with that rule will be made. The ALLOW_DERIVE option tells the Inference Engine that when the value of a variable is needed, backward chaining should be used to derive the value if possible.

This command will always run a system, and for some systems that should run rules in a particular order, such as smart questionnaires, it may be the best choice. However, for a pure backward chaining system, it is better to use a DERIVE command that will set "Goal" variables and use the Inference Engine to derive their value from the rules with backward chaining.

The RESULTS command displays a default results screen. This is just a screen that displays the value of all the variables set during the run with very simple formatting. It is fine for system development, but once the logic is working, this always should be replaced by a command to display only the results / conclusions relevant to the end user and with formatting to match a desired look-and-feel.

11.3 Command Block Window

To see a system's Command Block, click on the Command Block tab at the top, next to the "Logic Block" tab. The window shows the list of commands on the left, the command builder on the right and the command currently being built or edited on the bottom left.



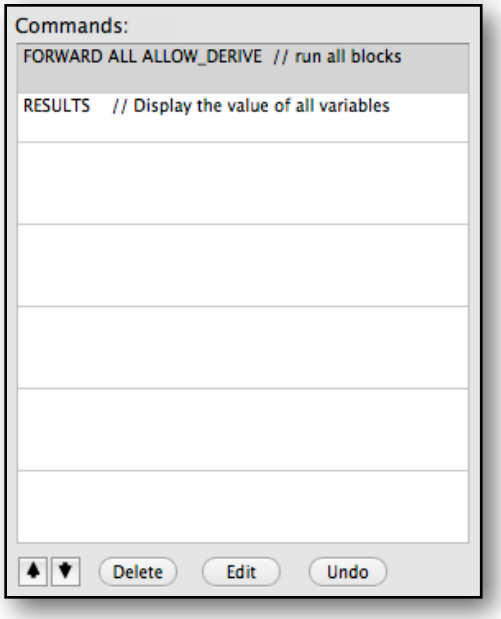
The Command List displays the commands in the Command Block. A command can be selected by clicking on it. This makes it the active command.

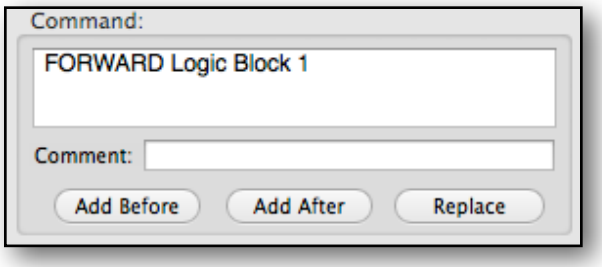
To edit a command, click the "Edit" button under the command list or double click on the command. The command will open in the command builder and can be edited or replaced.

To delete a command, click on it to select it and click the "Delete" button.

To change the order of commands, select a command and use the up and down arrows to move the command in the list.

To undo a change in the command list, click the "Undo" button.





The currently active command is displayed under the command builder in the “Command” edit box. The command can be directly edited in this window, but it is generally better and easier to use the controls in the Command Builder section of the window.

Commands can also have comments associated with them. The comments are optional and have no effect on the running of the system, but it is recommended that the reason for adding any command that is not obvious be included in a comment.

Once a command is built or edited, it can be added to the Command List. The command will be added relative to the currently selected command in the Command List.

- The new command can be added before (above) the current command by clicking the “Add Before” button.
- The new command can be added after (below) the current command by clicking the “Add After” button.
- The new command can replace the current command by clicking the “Replace” button.

11.4 Command Builder

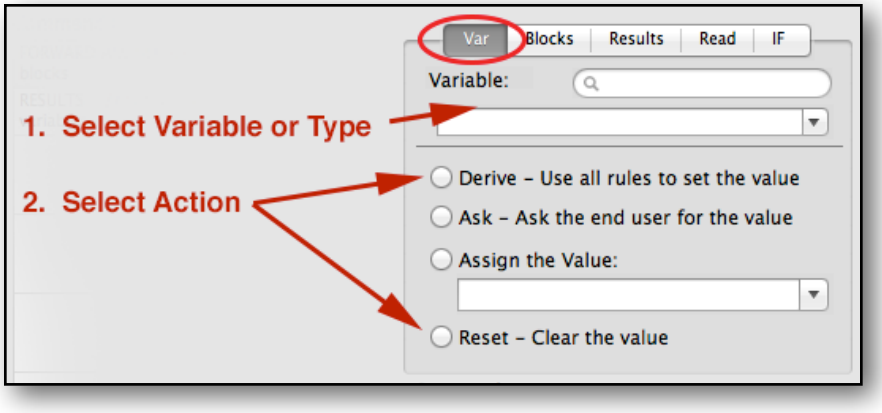
The command builder has 5 tabs for the different types of commands that can be built.

The tabs are:

VAR	Commands related to specific variables or groups of variables
BLOCK	Commands related to Logic Blocks
RESULTS	Commands displaying result screens and other types of screens
READ	Commands to read values from external files of data or programs
IF	Conditional IF and WHILE tests

11.5 VAR Commands

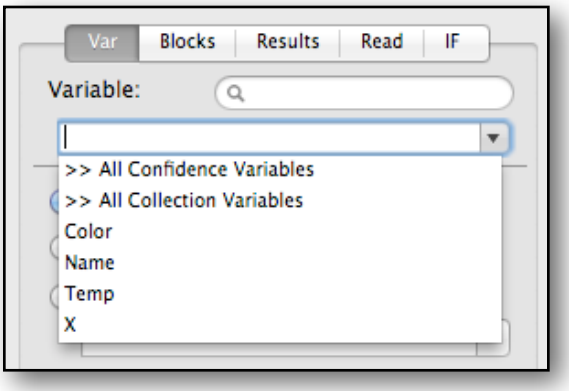
The VAR tab is for adding commands related to a specific variable, or a group of variables specified by variable Type.



Commands can be added to:

- Derive the value for a variable via backward chaining
- Ask the user for the value of a variable - regardless of the rules
- Assign a specific value to a variable
- Clear any value assigned (usually only needed when looping)

The first step in building a command on the VAR tab is to select the variable or group of variables by type. The drop-down at the top of the panel allows you to select any variable in the system or all Confidence or Collection variables by type. (Note the Types are limited to Confidence and Collection variables since these are the ones used to build most systems. Other variables cannot be selected by type and must be selected individually)



When “All Confidence” or “All Collection” variables is selected, the command is applied to all variables of that Type in order.

The next step is to select the action to take for the specified variable(s). Select the radio button next to the action to take.

Derive	Use **backward chaining** to derive the value of the variable(s) selected. All rules that set a value for that variable in their THEN conditions will be tested. When the IF conditions in a rule are true, a value will be assigned to the selected variable. (Note: If that rule also has other THEN conditions, those will also be applied possibly making assignments to other variables too). If no relevant rules exist, or none have true IF conditions, a value cannot be assigned from the rules and the end user will be asked to input a value for the variable. If the rules assign a value, the user will not be asked to input a value.
Ask	Immediately ask the user for the value for a variable. In this case, the end user will be asked to input the value, even if it could be assigned by rules. In most cases, if there are rules that can assign a value, those should be used via the DERIVE command, which will use the rules if possible and then ask. ASK is usually only used when a variable will eventually be asked due to the logic, but the desired user interaction is to have it asked at a particular point in the run. For example, a diagnostic system might eventually need to know the machine serial number, but that might be after many other questions have been asked. For purely user interface reasons, you may want to instead ask this at the start of the run, and the data will then be available later. Adding an ASK command at the top of the command block will force the variable to be asked at that point instead of later in the system. Unless a variable is sure to be asked by the logic, it is better to let the Inference Engine automatically ask for the value of variables as it needs them in testing rules or making assignments. That way, variables are only asked of the end user if they are required and cannot be derived.

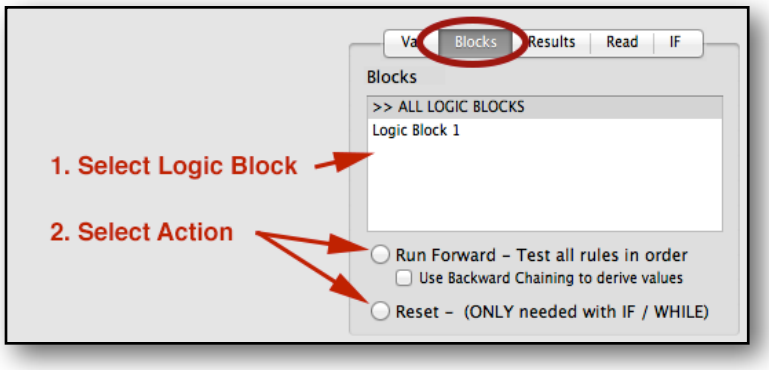
Assign	Assign a specific value to the variable. Depending on the variable's type, the box underneath will be either a drop-down list with the possible values for a list variable, or an edit box to enter the numeric or string value to assign. "Assign" can be used to set variables to specific values. For List variables, this will be a value from the variable's value list. For other variables, the value can be a simple number or string, or can be an expression that evaluates to a value that can be assigned to the variable. Any assignment that could be made in the THEN part of a rule can be used in an "assign" command.
Reset	Clear the current value of a variable. This is generally only needed with systems that use WHILE loops and is covered in the WHILE loop section 11.9.

11.6 BLOCK Commands

Commands related to Logic Blocks are added by clicking on the "Blocks" tab.

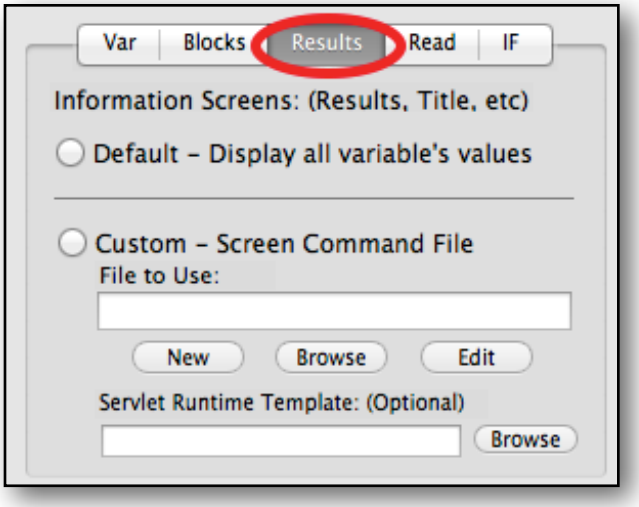
Select the specific block or "ALL LOGIC BLOCKS" in the list of blocks at the top of the panel. If "ALL" is selected, the action will be applied to each Logic Block in turn.

The next step is to select the action to take for the specified Logic Block(s). Select the radio button next to the action to take.



Run Forward	The Logic Block(s) specified will be run with **Forward Chaining**. The rules created by the Logic Block will be tested in order from the top down. If a rule's IF conditions are true, the THEN condition assignments from that block will be made. If the "Use Backward Chaining to derive values" option is selected, when the value of a variable is needed in running the rules, and that value is not already known, the Inference Engine will attempt to use backward chaining to derive the value of that variable from other rules in the system. The rules invoked via backward chaining can be anywhere in the system and are not limited to the specified Logic Block(s). If the "Use Backward Chaining to derive values" option is not selected, and a variable's value is needed, it will be asked of the end user.
Reset	Reset the rules associated with a Logic Block's "Unused". A specific rule will only "fire" once. (IF conditions tested and determined to be True or False, with THEN assignments made when the IF conditions are true) To have a rule reused more than once without restarting a system, it must be reset. This is generally only needed with systems that use WHILE loops and is covered in the WHILE loop section of this section 11.9.

11.7 RESULTS Commands

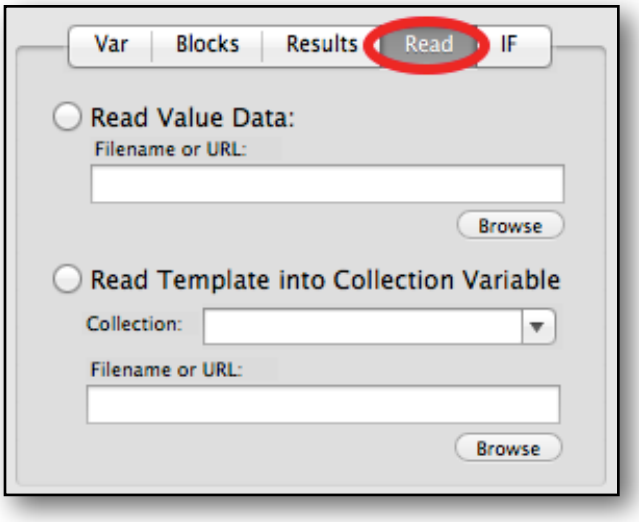


The Results tab is used to build commands that display results, reports or other information screens.

The top "Default" radio button is used for the default results screen. This displays the value for all variables set during the run. These are displayed in order and with very simple formatting. This is suitable only during system development to examine the variables and make sure the system ran as expected.

Since the default results screen is very simple, systems usually need a custom results screens. To add one select the "Custom" radio button and design the screen using the screen commands covered in the User Interface chapter 12. When running with the servlet runtime, a results screen can be designed with HTML and specified in the Servlet Runtime Template edit box. Running with the Exsys Servlet Runtime is covered in chapter 14.

11.8 READ Commands



The READ tab is for building two advanced commands. The READ command is for reading the value of variables from external files or URLs. This allows the Corvid system to be integrated with other programs and data sources. This can be used to set the values of variables, or since a URL can be called, it can be used call Java Servlets or other web programs that return data in the correct format.

The format for data to be read is:

[varname] value

where "varname" is the name of the variable in square brackets, and "value" is the value to assign.

The details of the syntax for each type of variable are covered in Appendix C.

A file or URL can set multiple variables by returning multiple variable/value pairs - one per line.

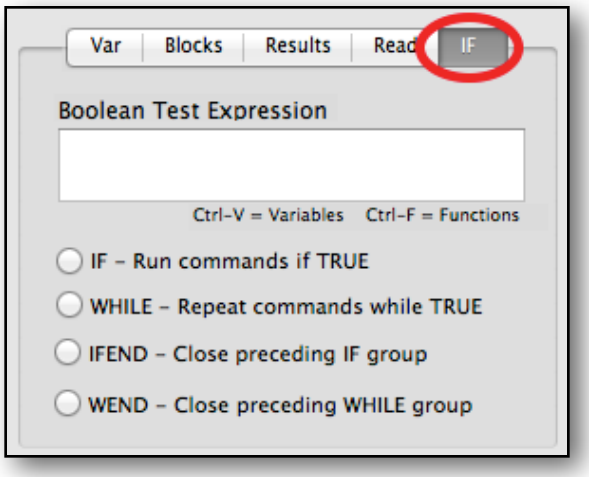
Also, the filename or URL can contain double square bracket embedded variables making it possible to have the Corvid system set the value of a variable that changes the name of the file or is a parameter passed to the URL. For example, a URL to a servlet might be:

http://myServer.com/myServlet/myClass?ID=[[ID_Var.value]]

In this case the value of the Corvid variable [ID_Var] would be embedded in the servlet call and passed to the called servlet. This could pass data to the servlet for it to use or to identify the data to return.

The second command on the Read tab is used to read a template into a Collection variable. This is a very quick way to build complex reports and is covered in the Reports chapter 15.

11.9 IF / WHILE Commands



The IF tab is used to build IF and WHILE tests so that sections of the command block will only be executed if, or while, a boolean test condition is met.

IF and IFEND commands are added to the command block to mark the end of the command section that the test condition applies to. The commands associated with the IF or WHILE will be indented.

IF and WHILE commands can be nested. An IFEND applies to first preceding IF command and a WEND applies to the first preceding WHILE command.

To Add an IF or WHILE group:

1. Select where to add the IF command by clicking on a node in the command list.
2. Build the IF (or WHILE) boolean test in the test expression edit box. This can be any Corvid expression that evaluates to TRUE or FALSE. As when building Boolean expressions in rules, Control-V will display a list of variables and control-F will display a list of functions to use in building the command.
3. Select the IF radio button or WHILE radio button for WHILE groups
4. Click the “Add Above” or “Add Below” button to add it to the command list relative to the command that was selected in step #1
5. **Click the IF (or WHILE) command just added to select it**
6. Click the IFEND (or WEND for WHILE groups) radio button
7. Click the “Add After” button.
8. Click the IF (or WHILE) command to select it again and build the first command to be in the IF (or WHILE) group.
9. Click “Add After” - this will put the command in the IF/WHILE group. It will be indented. Add other commands in the group relative to this command.
10. To add commands outside of the IF/WHILE group, select the IFEND/WEND, build commands and click the “Add After” button

(There are also various other ways to build IF and and WHILE groups. The key is that when IF and IFEND are added, the command between them are in that IF group and will be indented.)

Once commands have been entered they can be moved with the up and down arrow keys. The IF, IFEND, WHILE and WEND can also be moved with these keys as needed. IF and WHILE groups can be nested.

WHILE commands are built in exactly the same way as IF commands, but the commands in the group will repeat until the boolean test is false, and use a WEND command to close the group instead of the IFEND command.

WHILE Loops and Reset

WHILE commands allow creating loops in the command logic. The commands in the WHILE group will repeat until the boolean test is FALSE. This requires 2 things:

1. Something should change due to the commands in the WHILE group that will eventually make the boolean test FALSE. If this is not the case, it will be a loop that will repeat forever - and while that is legal, it is not usually a good idea and no commands past the WHILE group will be executed.
2. Since a rule is only tested and used one time, to reuse a rule requires using a RESET command to reset a Logic Block and often one or more variables.

If a Logic Block needs to be reused with new variable values, use the RESET command on the “Blocks” tab. This will reset the block so that the rules from that block can be reused. Likewise, to reset a variable, use the RESET command on the “Var” tab.

If a variable should get a new value each time the WHILE loop is executed, it should be reset. However, numeric, string and date variables may simply be overwritten by a new value, but generally it is best to reset them to force a new value to be set. Once a variable is reset, if it is needed, it will be derived from other rules or asked of the end user. (If it is to be derived, be sure to reset those rules too or they will only be used once)

For example, suppose a system asks the end user for the value of a numeric variable [X] and then should loop, asking the user for the value of [Y] until they enter a value of [Y] that is greater than [X].

This can be done with:

```
WHILE ( [X] <= [Y])
        RESET [Y]
WEND
```

This works by:

1. The test in the WHILE command will be evaluated. This means that, unless the values are already known, the system will ask the user for the values of [X] and [Y] to determine if the expression is TRUE or FALSE.
2. If the expression is FALSE, [Y] is greater than [X] (which is what is wanted) and the WHILE group commands will be skipped, and the next command after the WEND will be executed
3. If the expression is TRUE, the variable [Y] will be reset and the WHILE test reevaluated. Since [Y] has been reset, this will require asking the end user for the value of [Y]. If the end user enters a value less than or equal to [X], the loop will repeat. Note that [X] is not re-asked. The value of [X] was set the first time the WHILE test was evaluated and since it was not reset, its value is retained and continues to be used.

The same can be done with a Logic Blocks that do some calculation or analysis. Here we will use all the Logic Blocks in the system to derive the value of [Y]. This will require that, since we don’t know what specific rules will fire, all Logic Blocks be reset in each loop to make all the rules available.

```
WHILE ( [X] <= [Y])
        DERIVE [Y]
        RESET BLOCK=ALL
WEND
```

(In practice, this would probably require that some other variables also be reset or data read from an external file or URL with a READ command since otherwise the rules would always set the same value for [Y])

WHILE Loops to Implement FOR

Corvid does not have a FOR command as such, but the typical FOR command can be built using WHILE:

```
SET [X] 0
WHILE ( [X] <= 10)
        some commands
        some RESET commands - but do not reset [X]
        set [X]  ( [X] + 2)
WEND
```

This is equivalent to:

```
FOR [X] = 0 to 10 STEP 2
```

11.10 Keep the Command Block SHORT

Command blocks control how a system will run, but remember that command blocks typically should only be a few commands. Understand how the Inference Engine works and let it do the complicated work of finding, testing and using the rules that come from the Logic Blocks.

If you find yourself writing a program in the command block, you should rethink the problem to let the Inference Engine handle it with only a small number of commands and more in the Logic Blocks.

Most systems will not use WHILE unless they interface to other programs and even IF should only be used occasionally in the command block. If there are many nested IF commands in the Command Block, that logic probably should be in a Logic Block.

12 User Interface

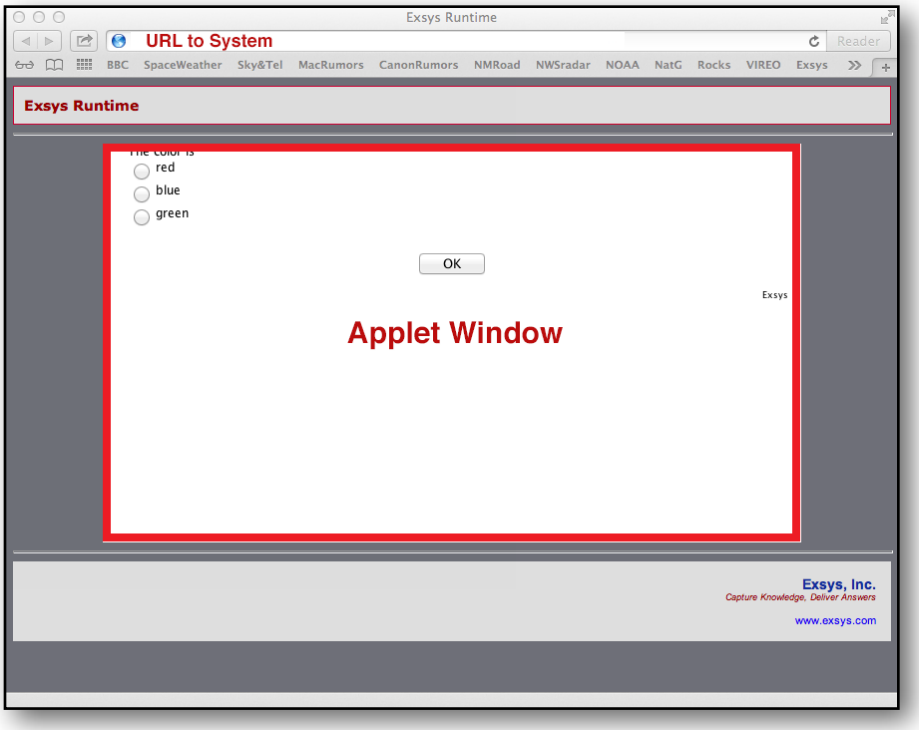
The user interface determines the look-and-feel of how questions will be asked and results displayed. This is controlled by setting various options in Corvid and defining results/report screens. These options control the fonts, colors, images, etc of the content in the applet window. When running with the Corvid Servlet Runtime, these setting are converted to CSS styles and used to control the look-and-feel of the content in the default HTML page. In that case, the CSS and page HTML can be edited to greatly extend the complexity of the end user interface. Running with the Corvid Servlet Runtime is covered in chapter 14.

There are 4 parts to designing the user interface for the Corvid Applet Runtime:

- The design of the HTML page outside of the Applet window on the page.
- Overall defaults for colors, fonts, etc that apply to all screens displayed by the Applet.
- Options set for individual variables that modify how those are asked or displayed.
- Designing screens that present information (Title, Results, Recommendations, etc).

12.1 HTML Page Outside of the Applet Window

When a system is run from Corvid with the Applet Runtime, an HTML page is automatically created named KBName.html, where “KBName” is the name of the system. This page combines a Corvid replaceable parameter in the template (defining the Applet Window where the system runs) in an overall HTML page that the browser can display.



Corvid builds this system HTML page using a default HTML template and then opens it with the Safari browser using the URL that points to the HTML page created for the system on the local computer.

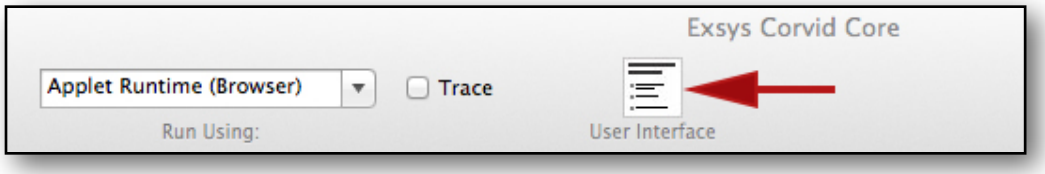
Changing the Design Outside of the Applet Window

The default template uses a simple design. All of the gray area outside of the Applet Window with “Exsys Runtime”, “Exsys Inc” etc, can be changed to any design or look-and-feel with an HTML editor. The HTML page can be modified to include Company information, logos, system title, links to related resources, etc. - anything to make the system fit into an existing web site or which the end user might need when running the system.

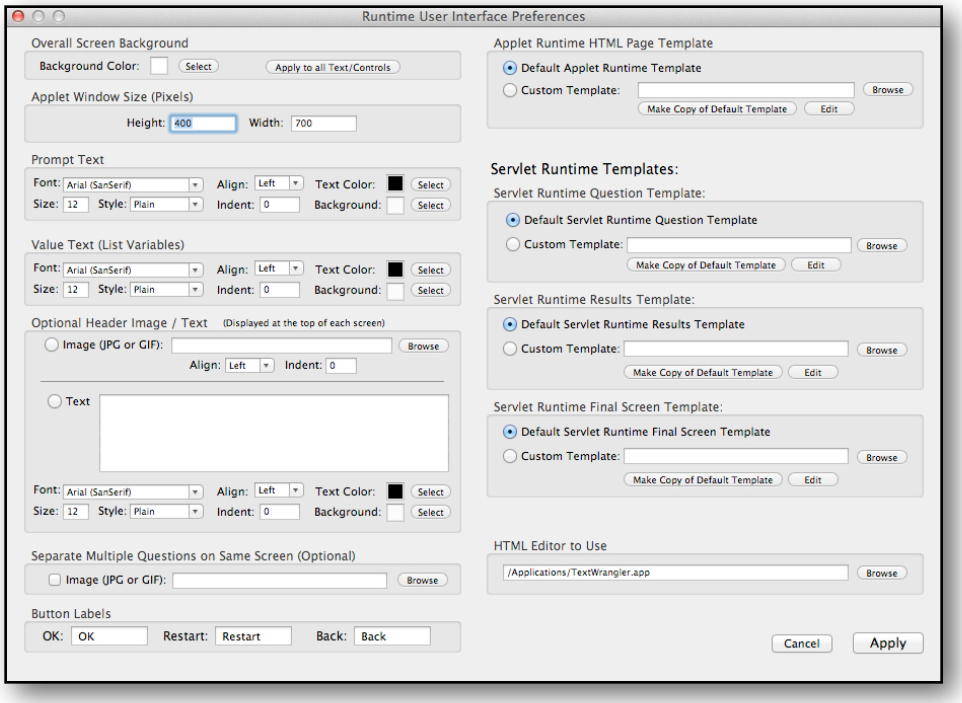
Generally changing the HTML outside of the applet window, combined with using colors and fonts in the applet that are consist with an overall design will make a Corvid system fit into an existing web site.

To do this.

1. Click on the “User Interface” icon at the top of the Corvid window.

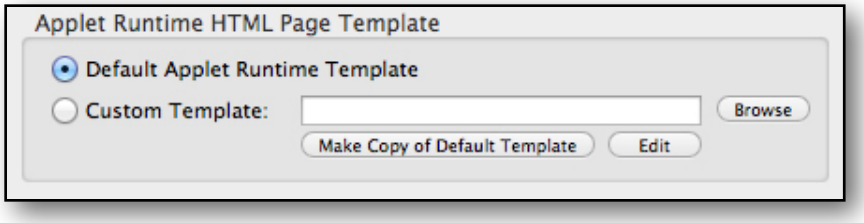


This will display the “Runtime User Interface Preferences” window.



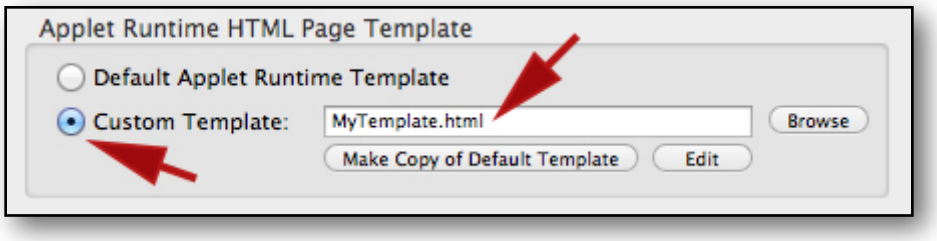
This window allows setting various options for content that appears in the applet window.

2. In the User Interface window that opens, in the “Applet Runtime HTML Page Template” section in the upper right, click on the “Make Copy of Default Template”.



Select a new name and location for the copy of the default template. Since this is a template, it can be located anywhere that is convenient. If you plan to always use this template in place of the default template, put it in a general "Corvid" folder that is convenient. The template can be put in the same folder as the other system files (.Corvid, .cvr, etc.), however, if it is put in that folder, it should **NOT be named *KBName*.html** (where KBName is the name of the system) since that is the system file that Corvid automatically generates from the template.

Corvid will select the radio button "Custom HTML Page" and copy the name of the new template in the edit box.



3. Use an HTML editor (Dreamweaver, etc) or a text editor that can edit HTML (TextMate, TextWrangler, etc) to open the copy of the default template that was created in step 2. (Note: The file cannot be edited with Apple's built in TextEdit unless its "Format" is set to "Plain text" in its "Preferences" window. Otherwise it will display the HTML design rather than allowing the code to be edited).

The default template file is standard HTML and CSS:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<title>Exsys Runtime</title>

<style type="text/css">


A:link {text-decoration: none}

A:visited {

      text-decoration: none;

      color: #003399;

}

A:active {text-decoration: none}

A:hover {text-decoration: underline}

.background {

      background-color: #6D6F77;

}
```

```
.Corvid {
	font-family: Verdana, Geneva, sans-serif;
	font-size: 14px;
	font-weight: bolder;
	color: #900;
	border: 1px solid #cc0033;
	padding: 10px;
	background-color: #ddd;
}
.ExsysAddress {
	font-family: Arial, Helvetica, sans-serif;
	font-size: 11px;
	color: #003399;
	text-align: right;
	padding: 15px;
	background-color: #ddd;
}
.CaptureKnowledge {
	color: #900;
	font-size: 9px;
	font-style: italic;
}
.ExsysName {
	font-size: 14px;
	font-weight: bold;
}
</style>
</head>

<body  class="background">
<div class="Corvid">Exsys Runtime</div>
<HR WIDTH="100%" SIZE="3">
<DIV align="center">
```

```
CORVID_RUNTIME_APPLET


</DIV>
<HR WIDTH="100%" SIZE="3">
<div class="ExsysAddress">
<span class="ExsysName">Exsys, Inc.</span><br />
<span class="CaptureKnowledge">Capture Knowledge, Deliver Answers</
span><br /><br />
<a href="http://www.exsys.com">www.exsys.com</a><br />
</div>
</body>
</html>
```

Any part of the file and CSS styles can be changed **except** the special line:

CORVID_RUNTIME_APPLET

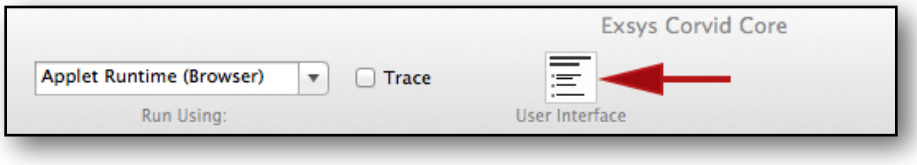
This line **MUST** remain in the template. Corvid will automatically replace this line with the actual applet tag using the system name, size options, etc. This should remain in the template as text. If it is removed or changed, Corvid will not be able to insert the applet tag and systems will not run.

Except for the "CORVID_RUNTIME_APPLET" text, anything else can be changed, deleted or modified. Any HTML / CSS / JavaScript, etc that is supported by a browser can be used. Corvid ignores and passes through anything else in the page except replacing the text "CORVID_RUNTIME_APPLET" with the applet tag. The template with the applet tag inserted will be the system HTML page and will be copied to KBName.html where "KBName" is the name of the system.

12.2 Setting System Default Fonts and Colors

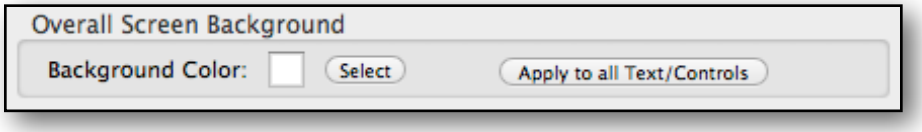
When running the system, the default colors and fonts can be set from the User Interface window. The setting will apply BOTH when running with the Corvid Applet **and Servlet** Runtime programs. .

Click on the "User Interface" icon:



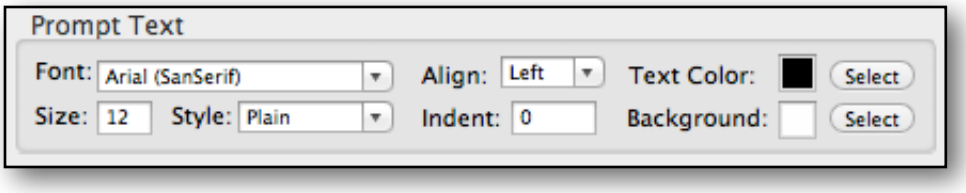
Background Color

The default color of the applet window is white. To set this to another color, click the "Select" button and choose a color from the standard color picker.



To easily set all of the text control to use this same color as their background color, click the "Apply to all Text / Controls" button. If this is not done, text and other controls can have contrasting background colors.

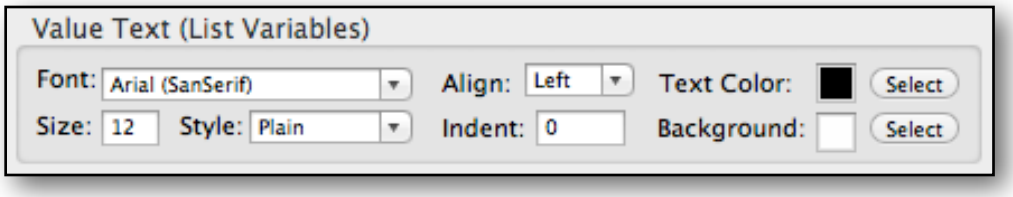
Prompt Text



The Prompt text is used when asking the end user for input, which will be inputting or selecting the value of a variable. The variable's prompt will be displayed to explain what is being asked. Setting a style for all prompts helps to create a consistent look-and-feel for the system regardless of what questions are asked.

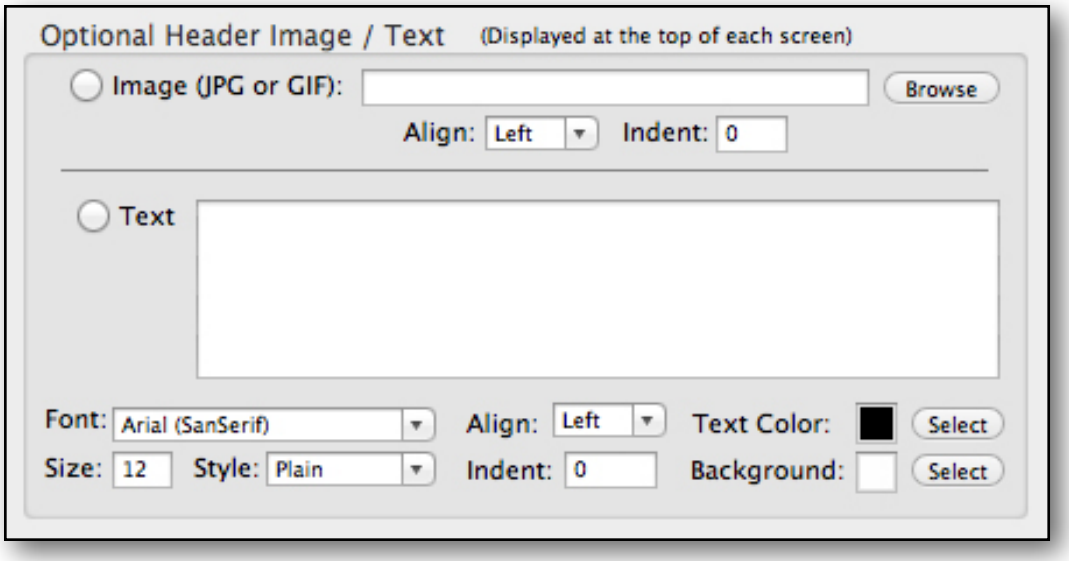
Select a font type, style, size, color, alignment and indent. This will be applied to all questions that the system asks.

Value Text



List variables will be asked of the end user by displaying the possible values for the user to select among. Just as with the Prompt text, the style of the values can be set to produce a consistent look-and-feel. This applies when the variable is asked using radio buttons or check boxes.

Optional Question Header



A system can optionally display an image or text at the top of each question screen. This is usually done to display a logo or system title. In most cases, this would be better displayed in the HTML outside of the applet window, especially since space in the applet window is more limited. However, when the applet window is part of a more complex page, it can be better to display the header in the applet window itself.

The header can be either an image or text.

For Images:

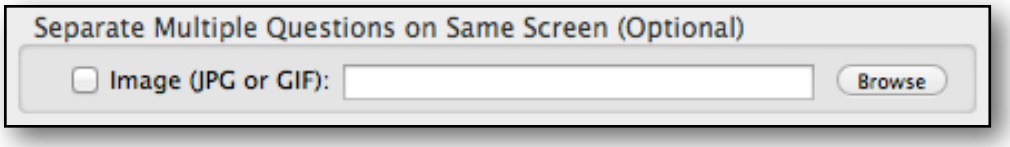
1. Use an image editing program to create a JPG or GIF no wider than the width set for the applet window.
2. Put the image in the same folder as the other files for the system (.Corvid, .cvr, etc).
3. Click the “Browse” button and select the file.
4. Select and alignment and indent to position the image in the applet window.

For Text:

1.Input the text to display.

2.Select the font, style, color and position for the text.

Also Ask - Multiple Questions on Same Screen

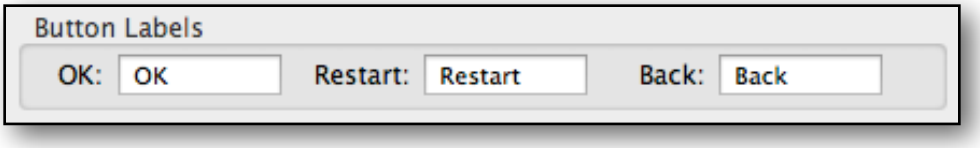


When a variable is asked, it is possible to have other variables asked on the same screen. The details for setting up screens to ask multiple variables is covered in section 12.3. However, when they are asked, an image can be displayed between questions as as divider. To do this:

1. Use an image editing program to create divider image. This should be a JPG or GIF no wider than the width set for the applet window, and generally around 10-50 pixels high.
2. Put the image in the same folder as the other files for the system (.Corvid, .cvr, etc).
3. Click the “Browse” button and select the file to use.

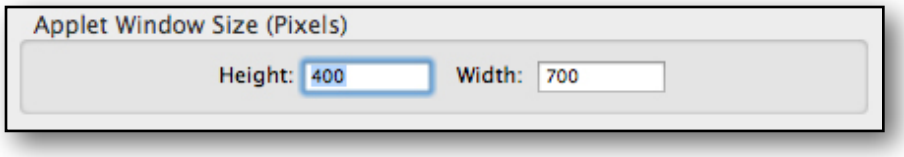
Whenever multiple questions are asked on the same screen, this image will automatically be put between questions.

Button Labels



When a system runs, it will have buttons for “OK”, “Restart” and “Back”. The default names for these buttons can be changed. This is usually not needed except when a site style guideline has specific button names, or a system is designed to run in a language other than English and the button labels should be in that language. To change the labels, just enter the new labels in the edit boxes. Corvid will use the new labels whenever the buttons are displayed.

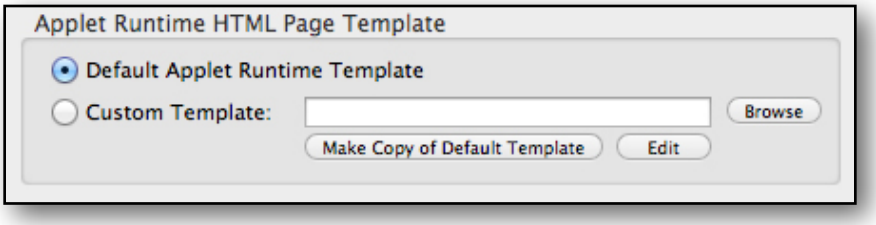
Applet Window Height and Width



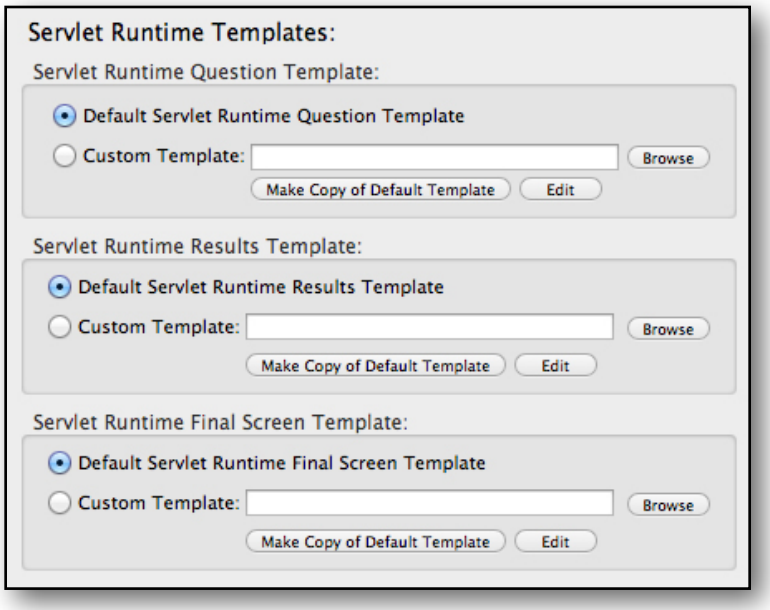
The height and width of the applet window can be set from within Corvid. These parameters will be passed through to the runtime program as part of the HTML applet tag. These should be set to match the overall system design. Generally it is better to have a height that allows questions to be fully displayed without the end user having to scroll the applet window - though that is certainly allowed when needed.

Template to Use

As discussed above, the HTML template can be selected and copied from the default template for editing.



Servlet Templates



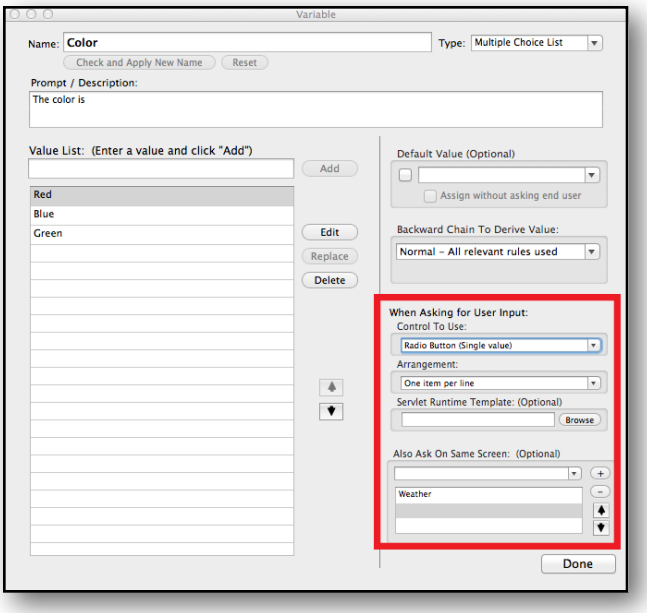
In addition to the applet properties, the templates to use when running with the Corvid Servlet Runtime can also be set. The default servlet templates will automatically pick up the styles, fonts and colors selected for the the applet window. Running with the Corvid Servlet Runtime is covered in chapter 14.

12.3 Options for Individual Variables

Each variable can have options set for:

- What controls will be use when the end user is asked to input a value for that variable.
- How those controls will be arranged.
- Other questions to ask on the same screen.

This can be done when the variable is created, or later by editing the variable. (In the Variables panel, select a variable and click “Edit”.)



In the window for the variable the lower right corner has the options for how the variable will be asked.

The options available depend on the type of variable. List variables have the most options since there are various ways to present the values to the end user. Numeric, String and Date variables are always asked with an edit box, so the only option is the size of the edit box and position relative to the prompt. Collection and Confidence variables are never directly asked of the end user and have no controls (However, other variables can be asked and then assigned to Collection or Confidence variables.)

List Variables

List variable can be asked using various controls depending on the desired user interface. The drop down list allows selecting Radio Buttons, Check Boxes, List boxes, Drop down list or Buttons. Radio buttons, Drop down lists and buttons should be used for variables that will only be assigned a SINGLE value by the end user. Check boxes and list boxes should be used for variables that may be assigned more than one value.

Radio Button	Single Value	The color is Red Blue Green
Check Box	Multiple Values	The color is Red Blue Green
List Box	Multiple Values	The color is Red Blue Green
Drop down List	Single Value	The color is Red
Buttons	Single Value No OK button	The color is Red Blue Green

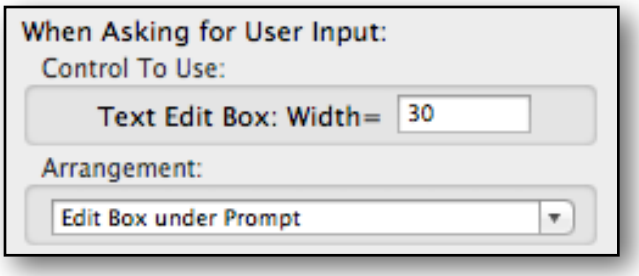
For a List variable with 3 values, the different options will create:

One per line below Prompt	
All in one line below Prompt	
All on same line as Prompt	

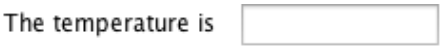
In addition the controls selected can be arranged relative to the Prompt text. The default is to put the controls below the Prompt with one control per line, but they can also be arranged all in one line or put on the same line as the Prompt.

Numeric, String and Date Variables

Numeric, String and Date variables are always asked with an edit box. The width of the edit box (in characters) can be set, along with the arrangement of the edit box relative to the Prompt.



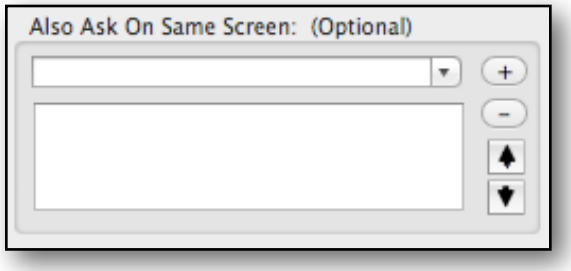
Text entered in an edit box will automatically scroll so any amount of text can be entered in any edit box, but if the nature of the question only requires a few characters be entered, it is better to use a small edit box. Likewise if a large amount of text needs to be entered, the edit box should be larger.

Below Prompt	
On same line as Prompt	

Also Ask List

Multiple variables can be asked on the same screen. This should only be done when:

1. The variables are related and make sense to ask together.
2. ALL of the variables will be needed, or will be needed if the “controlling” variable is asked. Also Ask should not be used to ask for the value of variables that will not be used in the system.



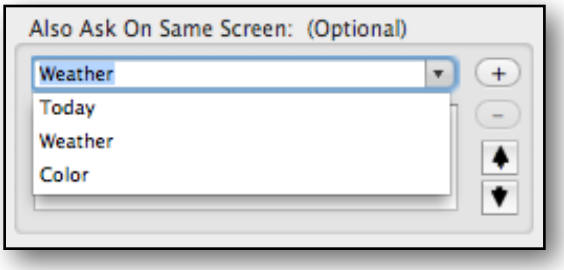
Multiple questions are selected by associating an “Also Ask” list with a “controlling” variable. When the end user is asked for the value of the controlling variable, the variables in its Also Ask list will be asked on the same screen. However, if a variable in the Also Ask list already has a value, it will not be included or re-asked.

1. Select the variable in the Variables list to add the Also Ask list to. This is the “controlling variable” and it must be asked to have the other variables also asked. Click the “Edit” button to open the window with that variable’’s properties window.

2. The lower right potion of the window has the Also Ask options. Click the drop down list to select a variable. This will list all of the variables that can be asked (List, Numeric, String and Date variables).

3. Select a variable and click the “+” button. The variable will be added to the list below.

4. Add as many variables as needed to the Also Ask list.

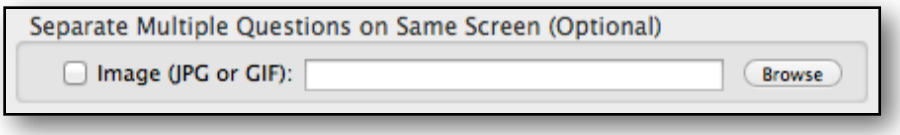


To remove a variable from the Also Ask list, click on it to select it and click the “-” button.

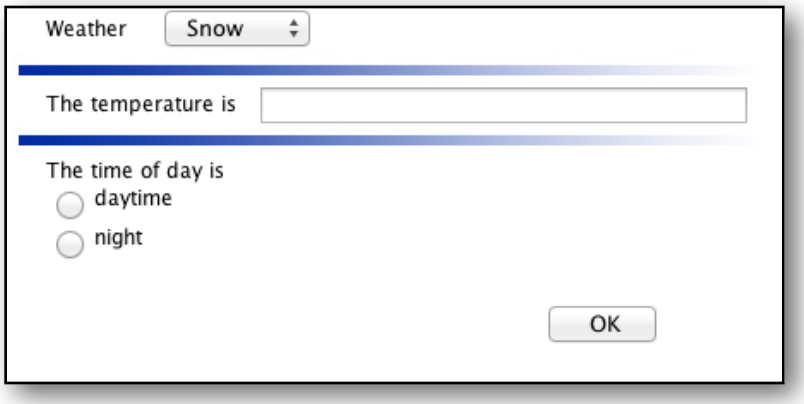
The controlling variable will be asked first on the screen, then the Also Ask variables will be added in the order that they appear in the list. To reorder the list, select a variable and click the up and down arrows to reorder the list.

Each variable will be asked with whatever controls are associated with that variable.

An image (JPG or GIF) can be added that will be placed between questions as a separator. This is done from the Runtime User Interface Properties window (click to the User Interface icon to display). The image file should no wider than the applet window and usually only 10-50 pixels high.



This is an example of a screen that asks for the value of 3 variables and uses an image between questions.



12.4 Screens that Present information

All screens other than question screens are built the same way. This includes:

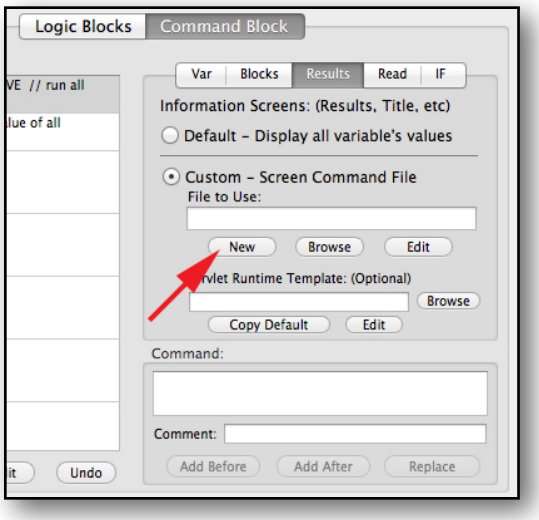
- Title Screens
- Result / Recommendation / Advice screens
- Other information screens

These are all “Custom Result” screens and are added in the Command Block.

Custom screens are built by creating a list of screen commands that each add an item to the screen. These can be text, variable values, images, etc. and can each be formatted independently. These are used to build the desired screen.

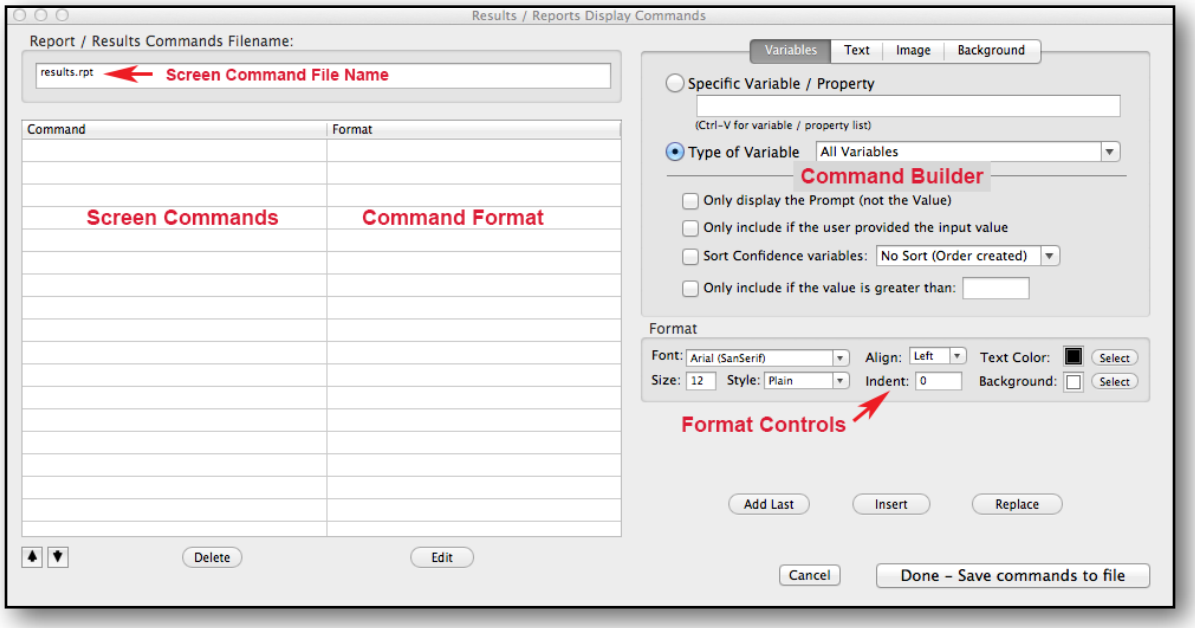
The point in the run where the screen will be displayed is controlled by the Command Block commands. For example, a title screen would be added at the top of the command block commands, a screen to display results would be at the end.

Go to the Command Block, select where the command should go and select the “Results” tab.



1. Click on the “New” button under the “Custom - Screen Command File” section.
2. The new screen command file must be given a name and location. The name should indicate what the contents will be, such as “title” or “results”. It is best to create it in the same folder as the other system files (.Corvid, .cvr, etc) Corvid will automatically give the file a .rpt extension.

3. This opens the Screen Command builder window:



Screens are created by a series of “Screen Commands” that each add some content to the screen (text, images, etc) sequentially. Each command has a command that specifies the content to add and a Format that sets the format options, which vary with the type of content.

The top left side of the window lists the file name. This is a text file with the screen commands. Below that is a list of the screen commands and their associated format. The right side of the window has the controls for building and editing the various commands and formats.

There are 4 tabs on the right for building the 4 types of content to add to the screen:

- **Variables** - The values of one or more variables set during a session.
- **Text** - Other text such as labels or static text. This text can also include double square bracket embedded variables.
- **Images** - Any JPG or GIF image.
- **Background** - For setting the background color of a screen and indicating the last screen in a system.

Below that is the Format section where format commands can be applied to the content.

Commands and their associated Formats are created and then added to the command list with the “Add Last”, “Insert” and “Replace” buttons. The new screen command is added relative to the currently selected command in the list.

The “Delete” “Edit” buttons under the screen command list allow commands in the list be deleted or edited in the command builder section.

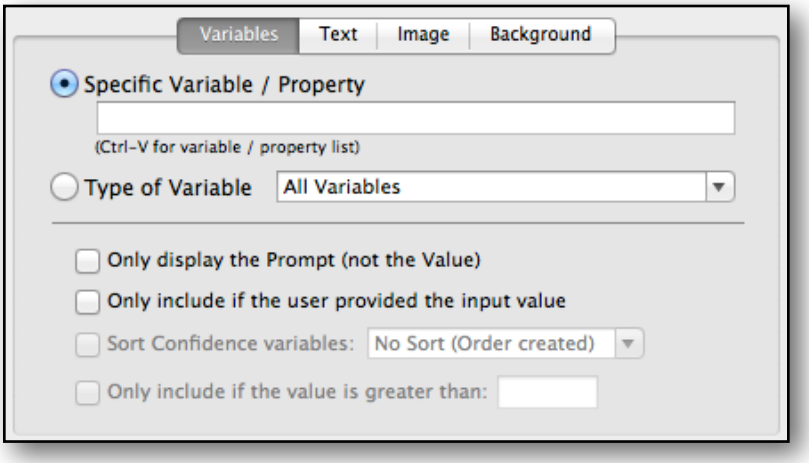
The individual lines in the screen command file can be moved up and down by selecting a line and clicking the up and down arrow buttons on the lower left corner of the window.

12.5 Variables

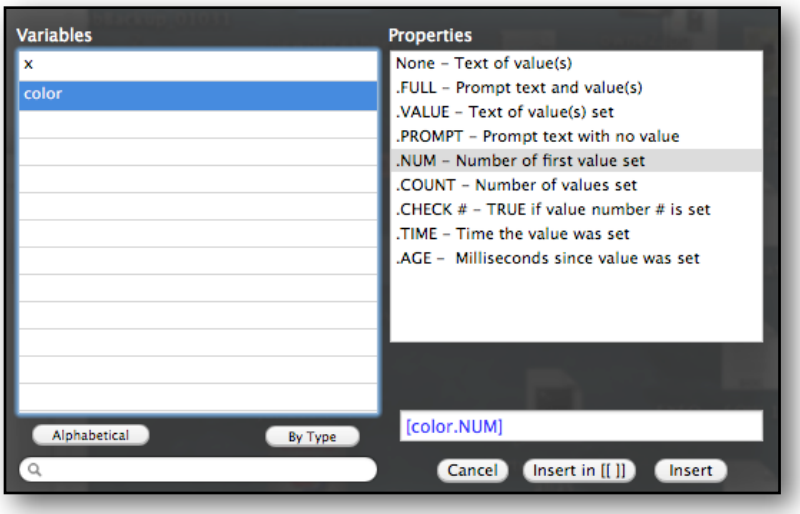
The “Variables” tab is used to add content to the screen that is the value or property of a variable or a group of variables by type.

This is often used to display the results and recommendations of a session in the final screen.

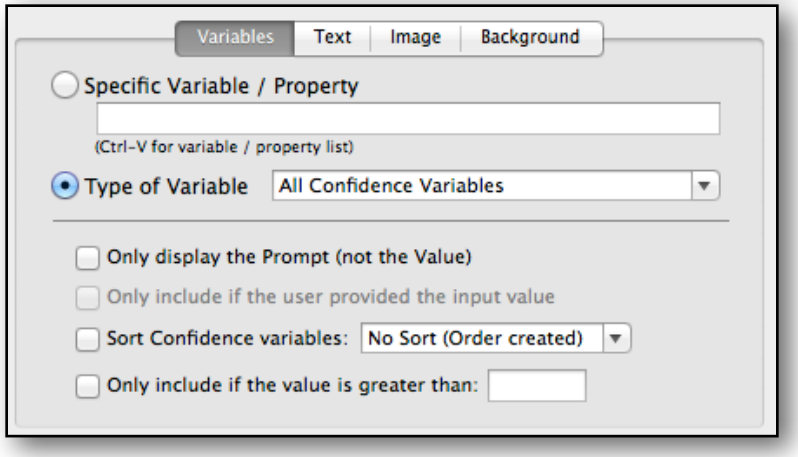
Select “Specific Variable / Property” to add the value of a single variable. Either enter the name of the variable in square brackets or the variable.property in square brackets.



In general, it is easier to select the variable / property from a list of variables. The variable list popup is displayed by pressing Control-V. Select the variable from the list and click “Insert”, or double click on the variable. To add a Property, select the variable and then the associated Property. If the Property requires other parameters, enter those and click “Insert”.



A group of variables can also be added by Type. Select the “Type of Variable” radio button and select the Type to display (Confidence, Collection, Numeric, etc). “All Variables” can also be selected from the drop down list. When a Type is selected, the variables will be added to the screen in the order they were added to the system unless one of the other ordering / limitation options is selected.



There are 4 options on the lower part of the “Variables” tab that sort or limit the output of the selected variable(s). These can be used in combination. Depending on the variable or Type selected, some of the options may be disabled since they would not apply.

Only display the Prompt (not the value)

This option is used only for Confidence variables. Many systems select among a group of Confidence variables to assign the most likely, or appropriate, ones a high value (confidence). This value can be used to sort the variables based on their score so that the most likely or appropriate ones are at the top. However, sometimes the actual value each variable was assigned may have little meaning to the end user. This option allows displaying the Confidence variable Prompt (which is the advice text), but without also displaying the value assigned. This option is generally used in combination with the “Sort Confidence Variable” option and often the threshold option.

In the case of all other types of variables, the value assigned is the information that would be presented to the end user the displaying only the Prompt would not be used.

Only include if the user provided the input value

This option is used in reports that echo back the input that the user provided. This is generally used with the “All Variables” option in the Type drop down to display all of the end user’s input values.

Sort Confidence Variables

This option is used only for Confidence variables. It is used to sort the list of Confidence variables in:

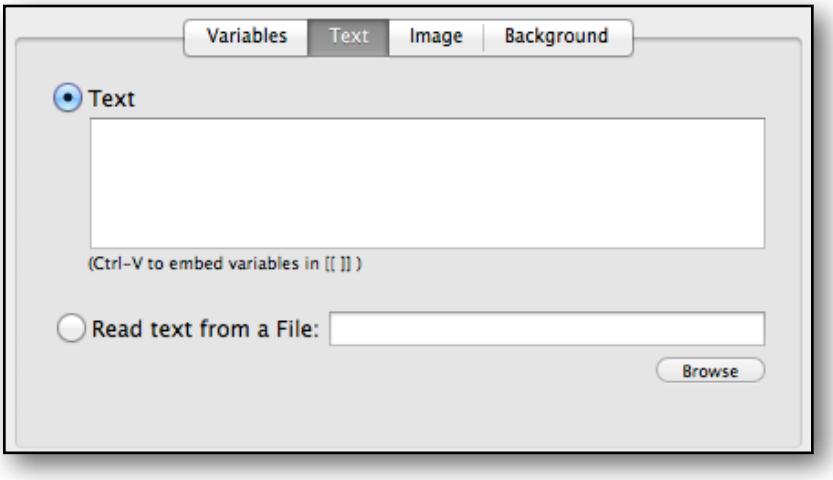
- Descending order - the ones with highest value at the top.
- Ascending order - the ones with the lowest value at the top.
- No Sort - variables listed in the order they were added to the system (this is the default).

This can be used with the “Only display Prompt” option to hide the actual values or the threshold option to limit a descending list to only those over a threshold value

Only include if value is greater than X (Threshold value)

This can be used to limit the display of a value by setting a threshold. This is generally used with Confidence variables to only display ones that are above a threshold value that indicates they are relevant to the end user because they got a final confidence value over the threshold.

12.6 Text



The “Text” tab is used to add text to the screen. This can be labels or other text content. Just enter the text in the edit box, or select to add the text from a file or URL by entering the file name or browsing to it.

If an external file is used for the text, it should be stored in the same folder as the system files or a sub folder.

The text or file contents can include double square bracket embedded variables / properties. In some cases when the results need to combine multiple variables, it is much easier to use double square bracket embedding to add values to a block of text.

For example:

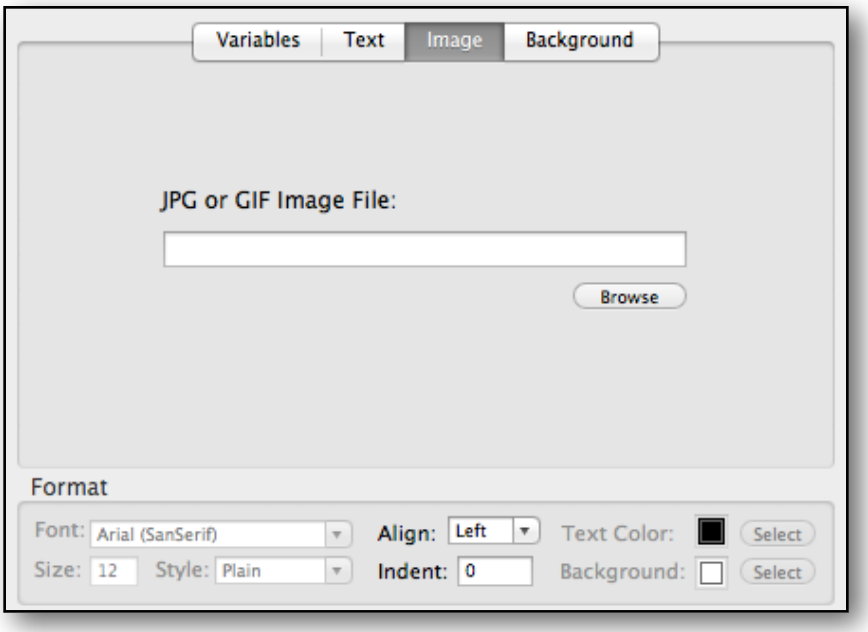
```
Since the temperature is [[temp.value]] degrees and the weather
is [[weather.value]], you should [[recommendation.value]]
```

This will fill in the values of [temp], [weather] and [recommendations] into a single text string.

When adding double square bracket embedded variables in the text, press Control-V to display the variable list. Once the variable / property is selected, click the “Insert in [[ ]]” button. Variables in the text that are in single brackets, rather than double square brackets, will not be replaced by their value.

Text can also include links and images. The HTML commands covered in section 12.11 can also be used in the text or file contents.

12.7 Image



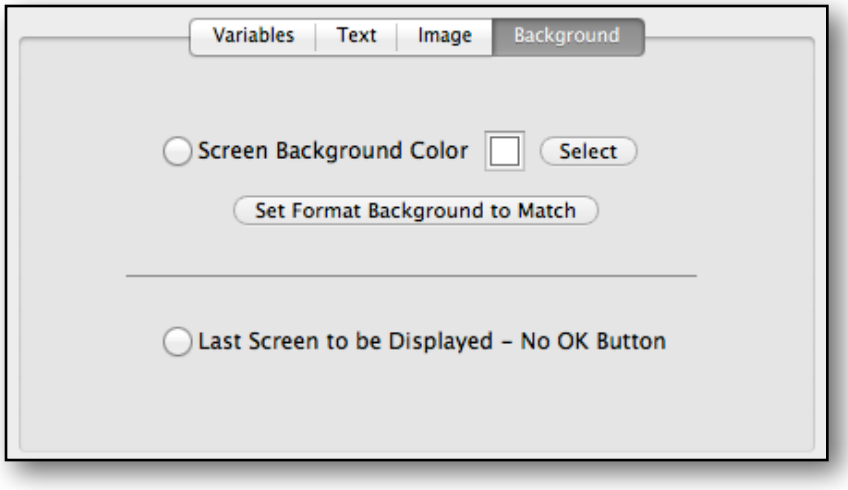
Images can be added to the screen from the Image tab.

Browse to the JPG or GIF file to display. This should be created at a pixel size that it will display well in the applet window. Image files should be in the same folder as the other system files or a sub folder.

Images have format options limited to alignment and indent.

The name of the image file can be set dynamically by making it a double square bracket embedded variable. For example, it the image name was set to [[selectedImage.value]], the system rules could dynamically select the image to display with the results.

12.8 Background

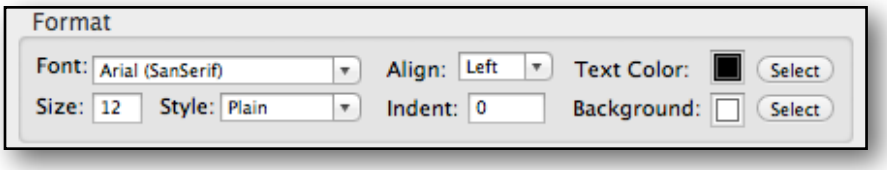


The background color of the screen can be set with the background command. The default is to have the background color be white. If some other color is preferred, the background command should be the first command in the screen command list. None of the format option apply to the background command.

Once the background color is set, the background color for text Formats can be set to the same value. This will make all text have that color for the background. To do this, click the “Set Format Background to Match” button. Text can have a contrasting background color, but usually screens look better if the background color matches.

The “Last Screen to be Displayed” command adds a command to the screen file that indicates it is the last screen displayed in the system and it should **not** have an OK button to continue on to another screen. With this option set, the screen will have a “Restart” and “Back” button, but no “OK” button. If this is not added to the last screen in the system, and the end of the command block is reached, a simple “System Done” screen will be displayed. Generally showing the “System Done” screen is bad design and can be easily prevented by adding a “Last Screen” command to the last system screen displayed.

12.9 Text Format



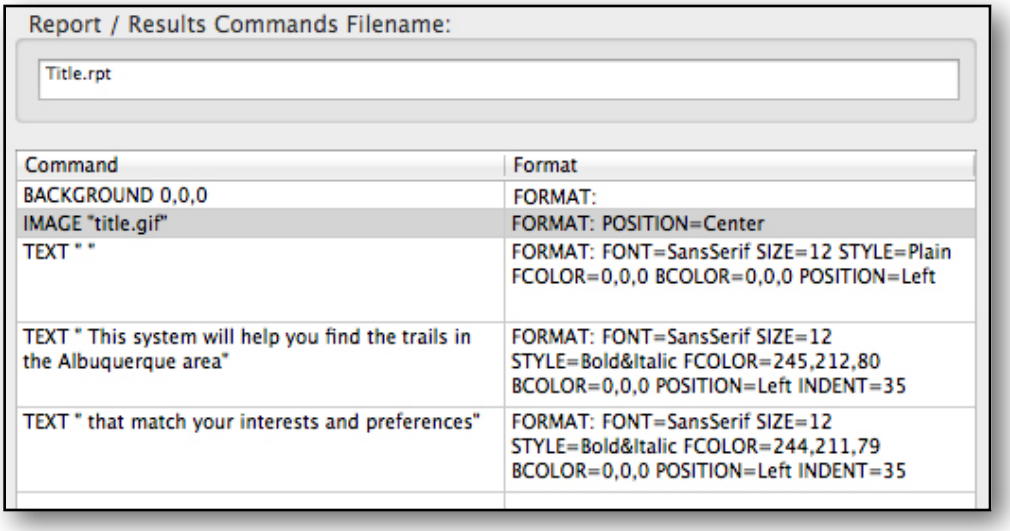
The Format options at the bottom of the window apply primarily to commands that add content in text form (Variables and Text commands). The format options are the standard ones for setting be font, size, style, alignment and indent. When setting a background color for the screen, it is generally best to set the Format “Background” to the same color. This can be done from the Background tab.

12.10 Examples of Custom Screens

Title and other Information Screen

A typical title screen might have a logo, title and brief explanation of what the system does. The easiest way to do this is to use a program like Photoshop to generate a .jpg or .gif file with most of the content. This allows images and more graphically interesting text to be added. Then put this in a custom screen. Other text not in the image can be added with other screen commands.

For example:



Command	Format
BACKGROUND 0,0,0	FORMAT:
IMAGE "title.gif"	FORMAT: POSITION=Center
TEXT " "	FORMAT: FONT=SansSerif SIZE=12 STYLE=Plain FCOLOR=0,0,0 BCOLOR=0,0,0 POSITION=Left
TEXT " This system will help you find the trails in the Albuquerque area"	FORMAT: FONT=SansSerif SIZE=12 STYLE=Bold&Italic FCOLOR=245,212,80 BCOLOR=0,0,0 POSITION=Left INDENT=35
TEXT " that match your interests and preferences"	FORMAT: FONT=SansSerif SIZE=12 STYLE=Bold&Italic FCOLOR=244,211,79 BCOLOR=0,0,0 POSITION=Left INDENT=35

The commands:

- Set the screen background to black (RGB - 0,0,0).
- Displays an image “title.gif” created in Photoshop.
- Adds a spacer line - a TEXT line with just a space in it.
- Add2 2 lines of additional text. This is set to have a black background and is indented. When adding text that goes over multiple lines, you can let Corvid wrap it automatically, but adding it one line at a time provides better control.

When displayed this looks like:

The gray area comes from the default Corvid template and could be edited as HTML. The black area is the applet window controlled by the screen commands. The OK button is added automatically so that the user can continue on in the system.

Other informational screens like this can be added anywhere they are needed in the system to provide the end user with information. They can be static screen like the title or include the value of variables more like the Results screen described below. Add commands in the Command Block to call them at the appropriate points in the system.

Simple Results/Recommendation Screen

A typical results screen might have an image, brief explanation of what the results mean, and the system recommendations.

The top of the screen is designed using the same techniques as the title screens, but the last item is typically either a command to display the Confidence variables that got the highest values or a Collection variable.

For example:

Report / Results Commands Filename:

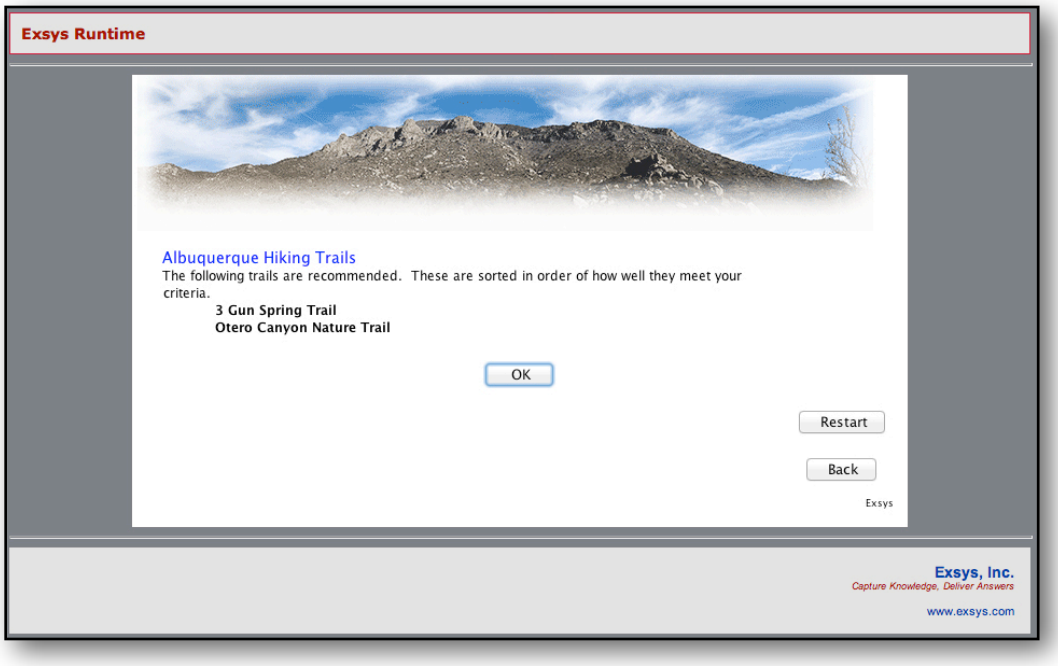
BestTrails.rpt

Command	Format
IMAGE "title2.gif"	FORMAT: POSITION=Left
TEXT " "	FORMAT: FONT=SansSerif SIZE=12 STYLE=Plain FCOLOR=0,0,0 BCOLOR=255,255,255 POSITION=Left
TEXT "Albuquerque Hiking Trails"	FORMAT: FONT=SansSerif SIZE=14 STYLE=Plain FCOLOR=0,0,255 BCOLOR=255,255,255 POSITION=Left INDENT=15 PROMPT_ONLY
TEXT "The following trails are recommended. These are sorted in order of how well they meet your criteria."	FORMAT: FONT=SansSerif SIZE=12 STYLE=Plain FCOLOR=0,0,0 BCOLOR=255,255,255 POSITION=Left INDENT=15 PROMPT_ONLY
Confidence SORT: DESCENDING LIMIT: "[#.VALUE] > 10"	FORMAT: FONT=SansSerif SIZE=12 STYLE=Bold FCOLOR=0,0,0 BCOLOR=255,255,255 POSITION=Left INDENT=20 PROMPT_ONLY

The commands:

- Display a header image.
- Adds a spacer line.
- Displays the title “Albuquerque Hiking Trails”.
- Displays a short explanation of the results that follow.
- Displays all Confidence variables that got a final value greater than 10 sorted “descending” (Highest values at the top).

When displayed this looks like:



12.11 HTML code in the Text

Any of the text in a system that is displayed to the end user can include certain HTML codes. When running with the Corvid Servlet Runtime, the user interface is HTML and displayed in a browser window, so all of HTML is supported and ANY HTML codes can be used. However, when running with the Corvid Applet Runtime, a few standard HTML tags are supported.

These HTML tags can be used to add links to any of the text displayed or to add images anywhere in the system. Even items like the values for a List variable can use be replaced by <img src=”..”> to have an image displayed in place of the text. This can be used to either display an image for the user to select, or to display an image that is text but formatted in a way more complex than the Applet Runtime allows.

Links

Any text that will be displayed can be marked as an HTML link with the standard <a tag:

```
<a href=”mylink.html”>text to link from</a>
```

Just wrap the text to link from with the and tags and the Corvid Applet Runtime will display the selected text as a URL where ever it is displayed. If the end user clicks on the link, it will display the specified URL in a new browser window or new tab (Depending on the end user’s browser settings).

The referenced HTML page can be relative to the system to CVR file, or can be anywhere on the web if the URL starts with “http://...”. If the HTML page is in the same folder as the other Corvid system files, just use the name of the HTML page.

Because of the way Java displays the links, it may put an extra space around it. If this is a problem, it can be compensated for by not adding spaces in the text around the linked word or phrase.

Since this uses standard HTML, the link in the text will work with both the Applet and Servlet Runtime programs. However, the servlet allows all of the additional options for the <a tag.

Links can be added to terms that the end user may not know, or to items of advice for more details. For example, the prompt text might include a word that the end user may not know. Making this a link gives them the option to get more details. A set of Confidence variables might generally describe various repair options, with links to HTML pages that provide the detailed instructions.

Images

Any text that will be displayed can include an image, or be replaced by an image, by using:

<img src=”MyImage.jpg”>

The image file can be relative to the other system files and in the same folder as the CVR file, or be anywhere on the web if the URL starts with “http://..” The supported image files are jpg and gif (including animated gif).

Images with Links

Images can also have links associated with them by using:

 <img src=”MyImage.jpg”>

Line Breaks

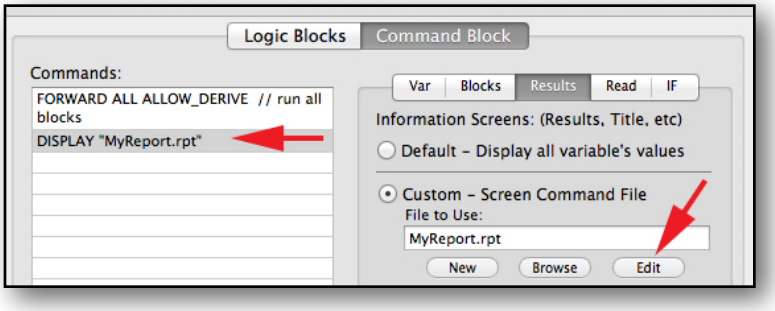
Any text that is displayed can include the HTML
 tag to add a line break.

12.12 Editing Existing Screen Command Files

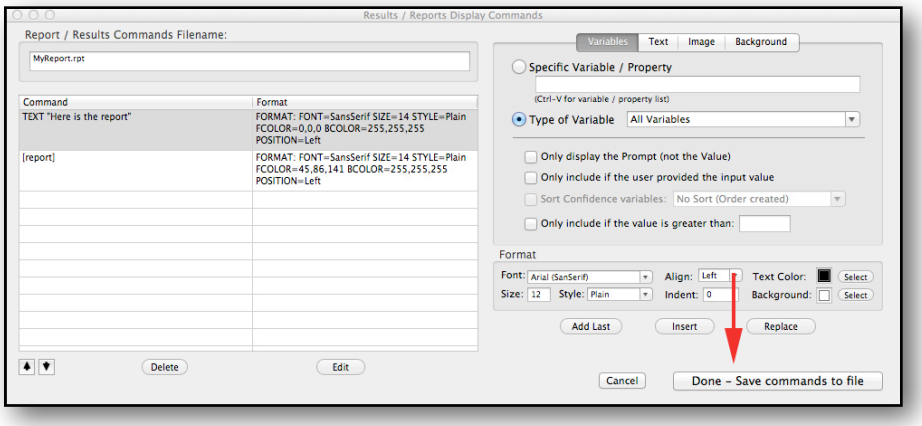
To edit the commands in a screen file:

Double click on the DISPLAY command in the command list. This will copy the file name to the command builder on the right.

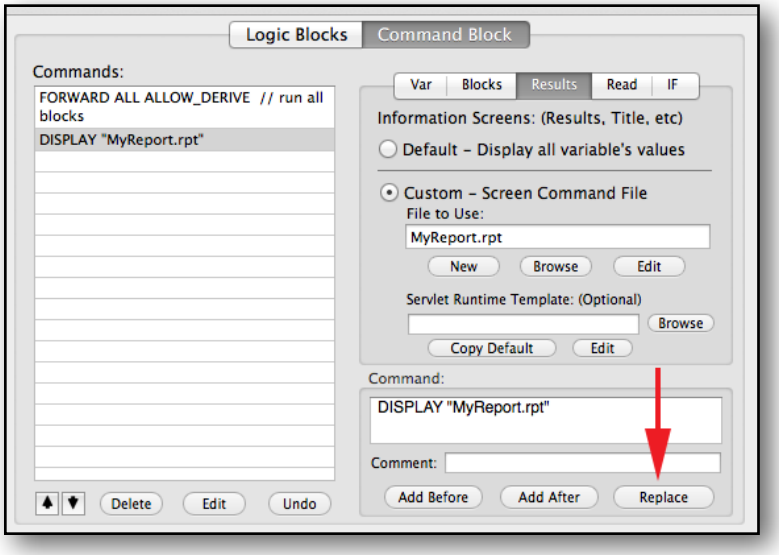
Click the Edit button to open the window to make changes to the commands in the file.



Make whatever changes are needed in the screen command, and click the “Done - Save Commands to File” button. Unless this is done, the commands are NOT saved to the file.



If the same file is still being used in the DISPLAY command, that is all that needs to be done. However, if the actual screen command file has changed, be sure to replace the command in the command file by building the new DISPLAY command and clicking the “Replace” button.



13 Running with Trace

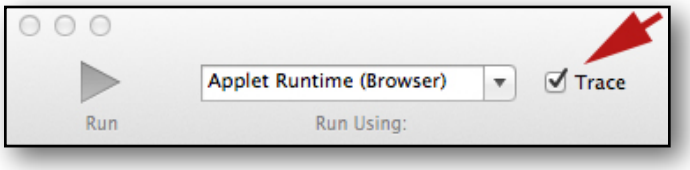
Occasionally when a system is run, it will ask an unexpected question or not come to the expected conclusion. Building Corvid rules is similar to programming, and like any type of programming, there are more ways to get it wrong than get it right. Corvid eliminates a lot of programming syntax and the Inference Engine does much of the work, but the logic in the rules still needs to be correct, and mistakes can be made.

Corvid provides a very useful Trace feature that allows you to look the variables and rules as a system runs to help find and correct any errors. It also displays the backward chaining goal stack, which can help in following complex backward chaining.

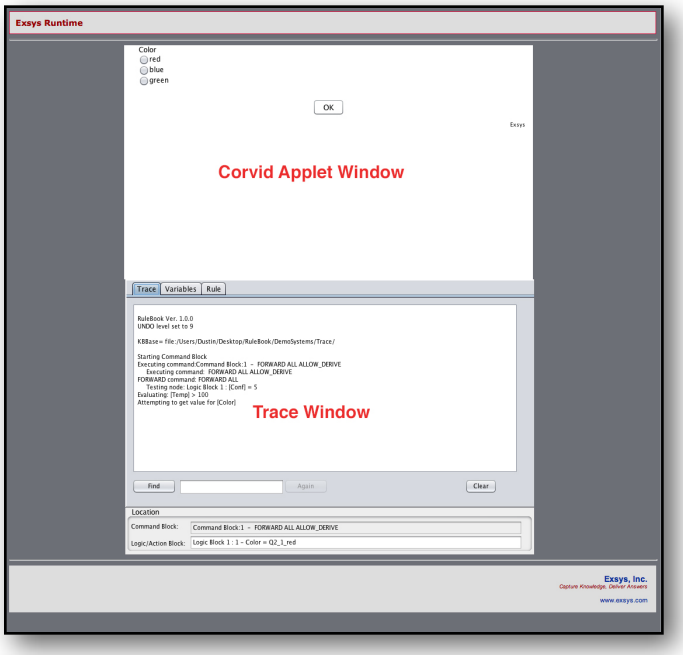
Corvid does a lot invisibly. In most cases, the developer just adds the rules and the Inference Engine will take care of the rest. If you want to see the details of what and how it is running, use Trace and it will provide many more details.

The full Trace function is interactive and only available when running with the Applet Runtime. This is one of the reasons it is generally best to get the logic of a system built first using the Applet Runtime and then add the user interface - which can be done in either the applet or servlet runtime. The Servlet does provide a more limited trace option equivalent to the first trace “history” panel.

To run with Trace select the Trace check box and run the system using the Exsys Applet Runtime. (This Trace check box does not apply to running with the Servlet Runtime.)



The Trace applet will be added below the normal Exsys Applet Runtime window.



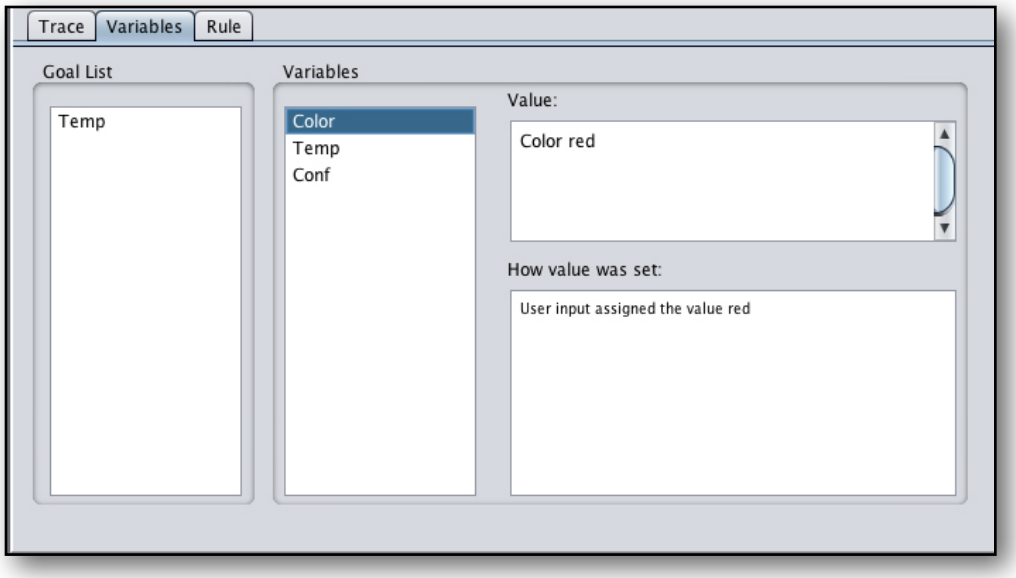
The Trace window has 3 tabs and a lower section that displays the current Command Block command being executed and the current rule being worked on by Logic Block name and row number and node being tested. List nodes values are displayed with the variable and value numbers, along with the value text.

Trace Tab

The “Trace” tab displays a detailed history of the session including which rules fired, where variables were set, etc. The “Find” box at the bottom allows searching for a specific variable or other text. As the system runs, additional text will be added to this window.

Variables Tab

The “Variables” tab allows examining the status of any of the system variables.



To see the status of a variable, click on it in the middle “Variables” list. The current value of the variable will be displayed to the right, along with details of how the value was set. This may indicate it came from the user, or was set by specific rules.

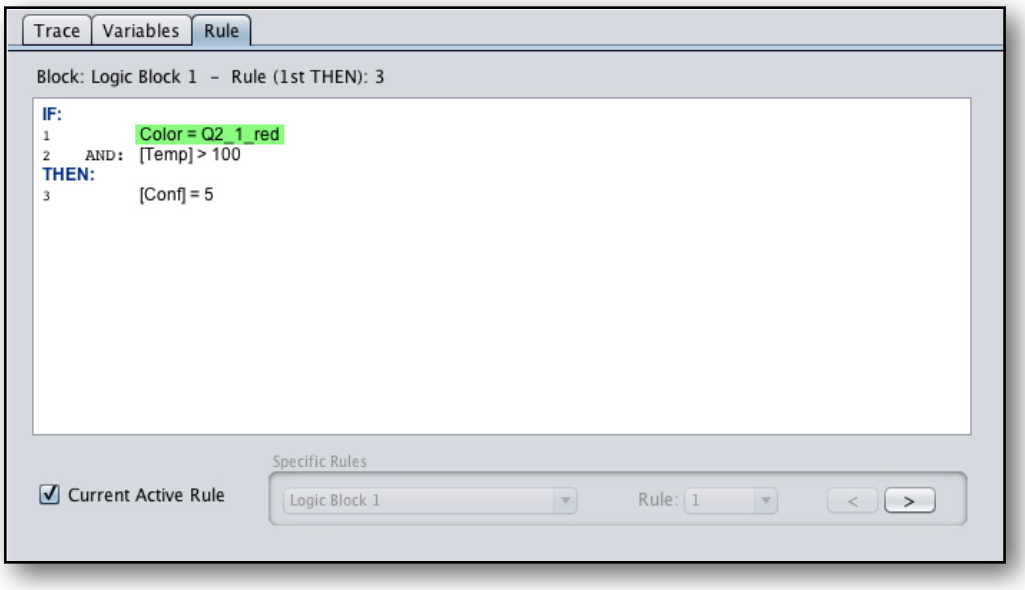
The left side of the Variable tab shows the backward chaining goal stack. The top variable in the stack is the one that is the current active Goal. It was added to the stack because it was needed to set the value for the next variable down in the stack, which is needed for the one below that, etc.

When the top level Goal (variable) has its value assigned, it is removed from the Goal Stack and the next one down becomes the new Goal. This may result in other variables being added to the stack. Variables are only removed from the top of the stack. When the stack is empty, that backward chaining task is done.

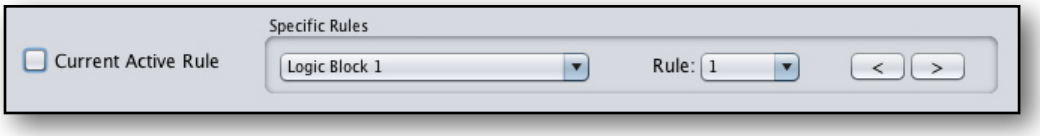
Rule Tab

The “Rule” tab displays the current rule being tested. IF conditions are color coded to indicate if they are True or False. Conditions highlighted in green are known to be true. Those highlighted in red are known to be false. If the rule is known to be false, the THEN conditions will be highlighted red. If the rule was true and fired the THEN conditions will be highlighted green.

The trace “Rule” tab automatically displays the rule currently being worked on. If the “Current Active Rule” checkbox is unselected, any rule in the system can be displayed and examined by Logic Block name and rule number in that block.



The Logic Block and rule (row in the Logic Block) can be selected from the drop down lists or can be stepped through by clicking the right and left arrow keys to move up and down in the rule list.



Trace in the Servlet Runtime

The trace applet window only applies when running with the Applet Runtime. However, the Corvid Servlet Runtime does allow a getting some trace information comparable to the “Trace” tab that displays the history of the session. This can be very useful, but is not interactive like the Variable and Rule tabs in the trace window. For information on using trace in the Servlet Runtime, see chapter 14.

Trace Did Not Start

The trace window uses inter-applet communication to get data from the main Corvid applet window. Occasionally due to system load or other factors, the trace window may not start quickly enough and does not get connected. If this happens and the trace window is blank, click the Reload icon in Safari to reload the windows. At that point they are in the cache and should load without any problem.

14 Using the Corvid Servlet Runtime

Along with running systems with a Corvid Applet Runtime, systems can be run using the Corvid Servlet Runtime. The servlet runtime has many advantage such as:

- Far more complex end user interfaces designed with HTML
- Running systems on devices that do not support Java applets (e,g, iPhone, iPad)
- Eliminating any requirement that the end user browser have Java or allow Java applets to run.

Corvid is designed to make creating systems for the Corvid Servlet Runtime easy for beginners, but with the flexibility to create complex interfaces when needed.

14.1 How the Corvid Servlet Runtime Works

The Corvid Servlet Runtime is a Java Servlet that is run under Apache Tomcat. It runs a Corvid system by sending HTML forms to the end user's browser. These forms may ask a question or present data. The end user interacts with controls on the form to answer questions and sends the input back to the Corvid Servlet Runtime on the server. This adds to the data the system has available which, based on the system logic and rules, may lead to another question or the display of results. The question sequence and analysis is the same as when running with the Corvid Applet Runtime, it is just the technique that the runtime uses to interact with the end user that is different.

Since the Corvid Servlet Runtime uses HTML forms, rather than a Java Applet, it will work on any browser regardless of Java support. This means Corvid systems will run on any browser including iPhones and iPads.

One of the many things that Apache Tomcat does is keep track of each user's data and session. This allows many users to simultaneously be running the system on the server without overwriting each others data. Corvid automatically embeds data into the forms to identify the session the data applies to.

The forms that the Corvid Servlet Runtime uses to ask the end user questions and present results are based on HTML "Template" files that define the look-and-feel of the system. Corvid has built in default templates that display the same font/style and UI command settings as the applet, but converted into HTML and CSS. The default templates are very easy to use since they pick up the basic settings used for the applet. A system can be developed with the Corvid Applet Runtime and then run with the Corvid Servlet Runtime with the same user interaction and a very similar look-and-feel. This makes it very easy to move systems to the Corvid Servlet Runtime.

For more control and complex user interfaces, the Corvid default templates can be edited to customize them. The templates use parameters that Corvid automatically fills based on system data. These can be combined with as much, or as little, HTML as needed to create the desired user interface. Corvid makes it easy to switch from the default templates to customized one.

14.2 Install and Configure Apache Tomcat

Using the servlet runtime requires a little more setup on the development machine and requires fielding the system on a server with a "Servlet Container" such as Apache Tomcat, Glassfish, IBM Websphere, etc. Corvid is designed to integrate with Tomcat, which is recommended at least for the development environment. Completed systems can be moved to other servlet containers on production servers running any operating system.

Developing systems for the Corvid Servlet Runtime it is much easier if the local Mac has the ability to do the use the same edit/test/edit approach as when building for the Applet Runtime. This requires that Apache Tomcat be installed on the local Mac. At one point, Tomcat was a part of OSX and automatically came with the Mac operating system, but recent versions of OSX have not included it. Fortunately, Tomcat is a free download and easily installed. Once it is installed, it can be configured from within Corvid, including testing, starting and stopping Tomcat. Tomcat and the Corvid servlet are easy to install on a Mac and Corvid does many things automatically to make it easy to install and test. Just follow the instructions below exactly, and in a few minutes Tomcat should be running.

Install Apache Tomcat

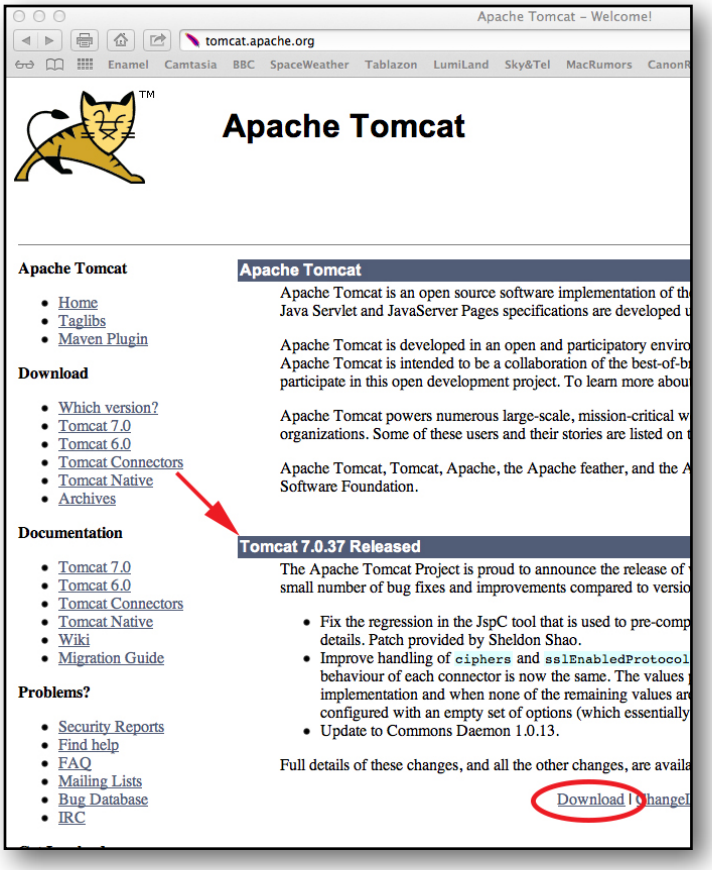
1. Tomcat does effectively give your computer some aspects of a server while Tomcat is running. Check with your corporate IT group to see if there are any special requirements or restrictions on installing Apache Tomcat.

2. Open Safari and goto:

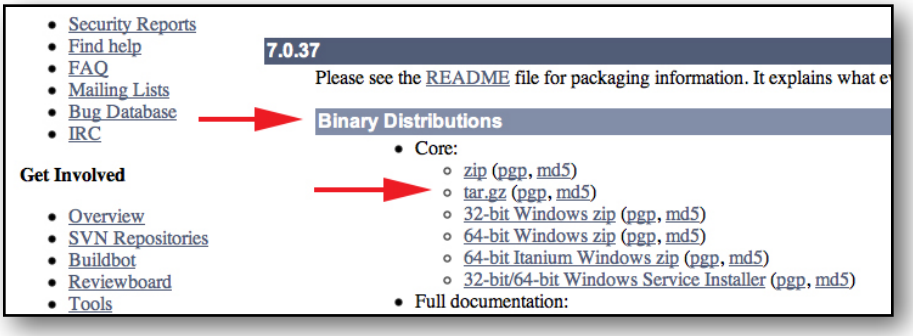
   http://tomcat.apache.org

3. Select either Tomcat 7 or Tomcat 6 and click the “Download” link.

   Either 6 or 7 will work. Some organizations may not have switched to Tomcat 7 and have policies requiring Tomcat 6. If there are no restriction, Tomcat 7 is recommended since it is the latest version.



4. Go down to the “Binary Distributions” section of the page and double click on “tar.gz” This will download a binary that will work on the Mac, does everything Corvid needs and is the easiest to install.



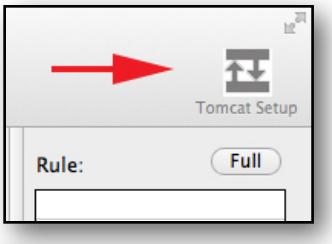
5. A file will be downloaded named “apache-tomcat-7.xx.xx.tar.gz” where the “xx” will be the specific current version number.

6. Once the file has finished downloading, go to your “Downloads” folder, find the file and double click on it. This will expand the tar file into a **folder** named “apache-tomcat-t.xx.xx”

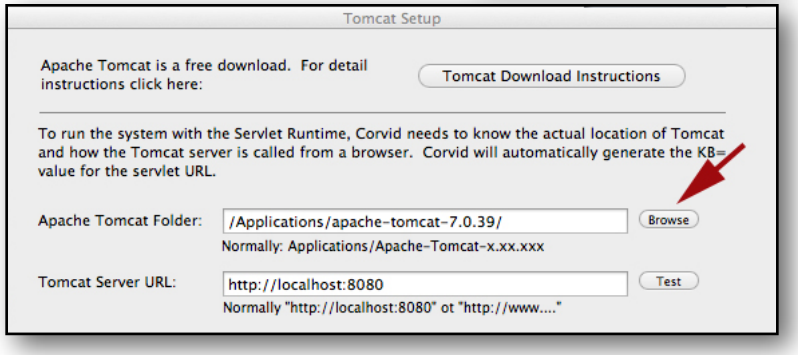
7. Move this **folder** to a where you want to keep Tomcat. The 2 best places are your Applications folder or your User folder. Either will work. Just drag the entire folder to the location selected. Remember where the folder was placed, you’ll need it to configure Tomcat in Corvid. The folder must be moved and should not just be left in the “Download” folder.

Configure Apache Tomcat in Corvid

8. Go back into the Corvid window and click on the “Tomcat Setup” icon in the upper right.

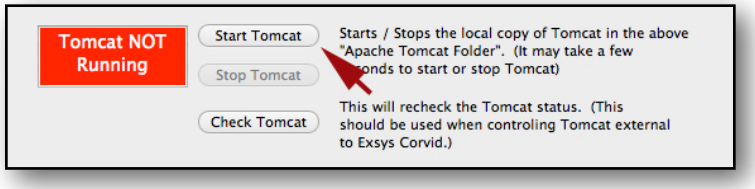


9. In the Tomcat Setup window, goto the “Apache Tomcat Folder” section near the top and click on the “Browse” button. Go to where the Tomcat folder was placed and select the **folder**. (Note this is the **folder**, not a file in the folder). The edit box should display “yourPath/apache-tomcat-7.xx.xx./”. Where “yourPath” will be “Applications”, your User folder or wherever the Tomcat folder was placed in step #7.

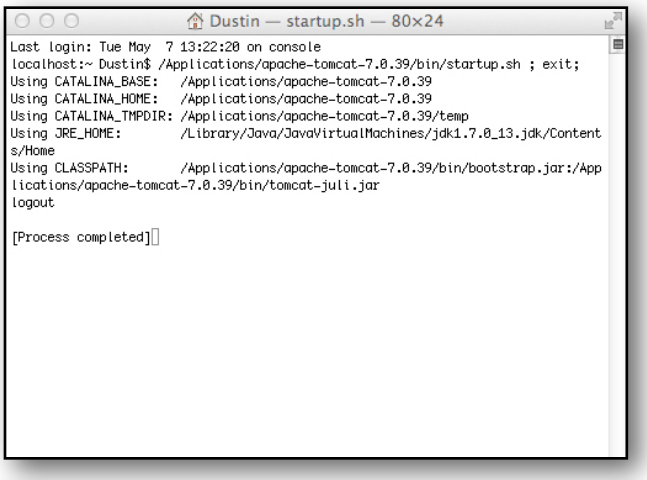


10. By default the Tomcat URL is “http://localhost:8080” this will be automatically filled in in the Tomcat Server URL box. This should not need to be changed unless your computer already had Tomcat installed with other than the default options. If that is the case, you may need to change the URL to whatever was set for the install. If in doubt, contact your IT group. This URL should be for your local computer, not a copy of Tomcat installed on a web server. Corvid only has the ability to control Tomcat and generate paths when using Tomcat locally. System can eventually be moved to Tomcat web servers, but development is much easier if done locally.

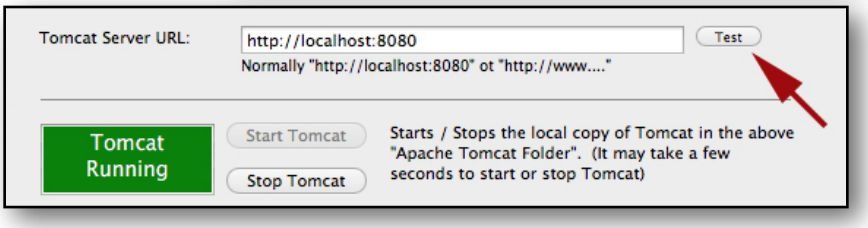
11. Goto the red “Tomcat Not Running” box and click the “Start Tomcat” button. The red “not Running” should change to a black “Waiting..” message. In a few moments, the red box should change to a green “Tomcat Running” box. Depending on how fast your system starts up Tomcat, the “Waiting..” message may return to a red “Not Running” message. If this happens, you may need to press the “Start Tomcat” button again and it should change to the green “Tomcat Running” message. If it does not show the green “Running” message after clicking the “Start Tomcat” button 2 or 3 times, there was a problem with one of the preceding steps. Check the steps for any problem. (If Tomcat was already installed for other reasons, and is already running, click the “Check Tomcat” button and see if the box turns green.) If Tomcat is not starting, this must be corrected before continuing. Check with your IT group.



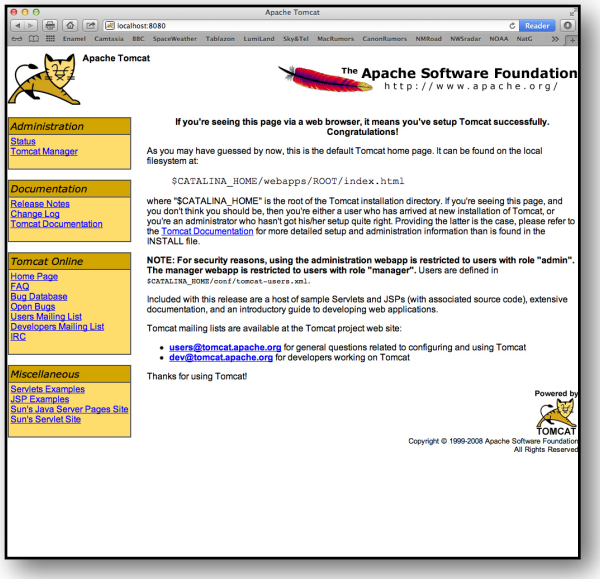
When Tomcat is started or stopped,a Console window will appear somewhere on your screen displaying the commands used to start Tomcat. This window can be ignored or closed. If there is any problem starting Tomcat, this window should display information on the problem. When done with Corvid, just close any Console windows remaining.



12. Once Tomcat is displaying the green "Tomcat Running" message, click the "Test" button next to the "Tomcat Server URL" edit box.



This should display the Tomcat title window in Safari. If the Tomcat title window was not displayed and a "Cannot connect" error is displayed, Tomcat is not running or the "Tomcat Server URL" edit box is not correct. The default "http://localhost:8080" should be correct unless Tomcat was already installed and configured differently. In that case, enter the URL configured for Tomcat or contact your IT group.

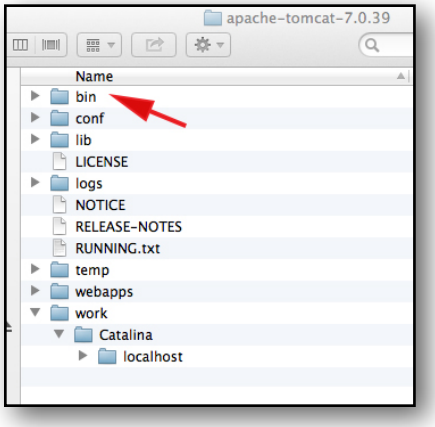


Close the Safari window with the Tomcat title screen window.

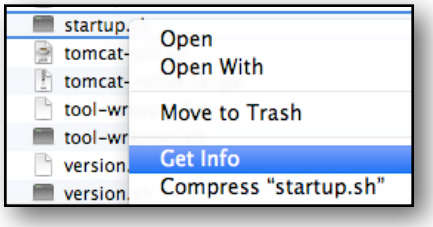
If clicking the "Start Tomcat" button did not start Tomcat:

Depending on system settings, the "Start Tomcat" and "Stop Tomcat" buttons may not initially work. To correct this:

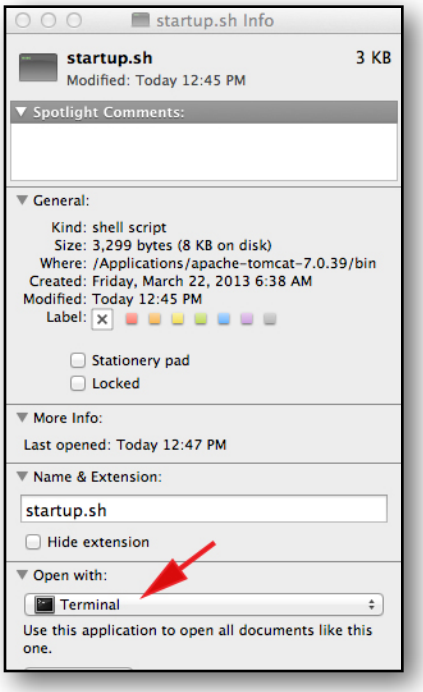
- Find the Apache Tomcat folder in Applications and open the **bin** folder.



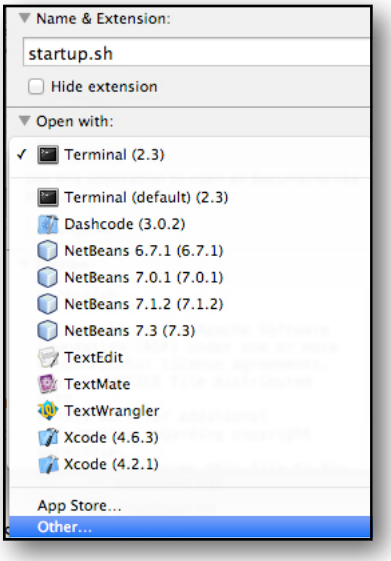
- In bin, find the file startup.sh (**Note:** there is also a startup.bat - you want the .sh file)
- Right click on the startup.sh file and select “Get Info”.



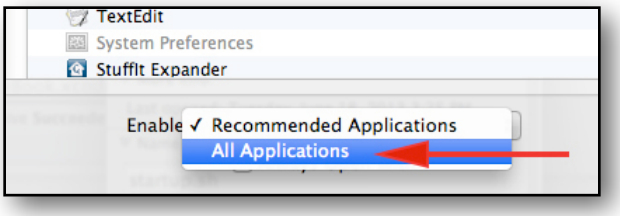
- Go to the “Open With” section. It should be set to “Terminal”, if it is not:



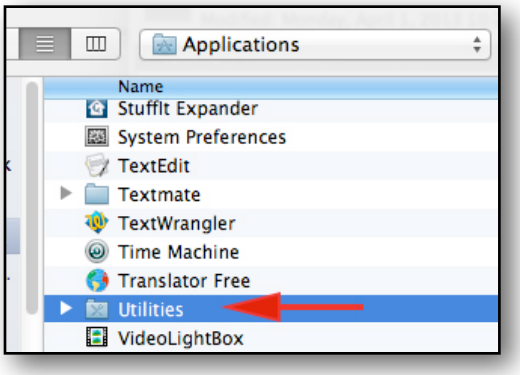
- Click on the drop down list under “Open With” and select “Other” at the bottom.



- At the bottom of the file selection window displayed, change “Recommended Applications” to “All Applications” by clicking on the drop down.

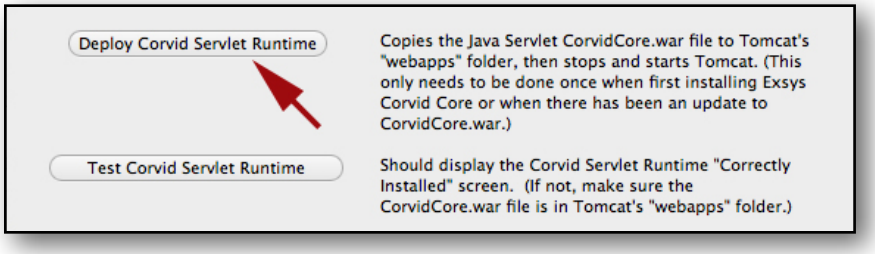


- Go to your Applications folder and open the “Utilities” folder.



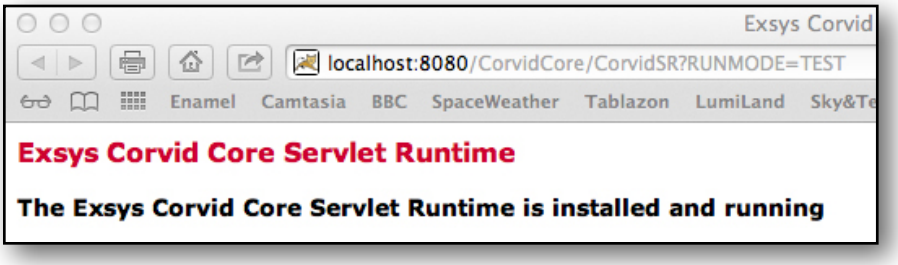
Deploy the Corvid Servlet Runtime

13. Click the “Deploy Corvid Servlet Runtime” button. This will copy the Corvid Servlet Runtime from the Corvid app to the Tomcat folder. This only needs to be done ONCE when Tomcat is first installed or when a new copy of the Corvid Servlet is provided with an update to Corvid. It will not cause any problem to do it more than once, but it is not necessary. You should see a window saying the servlet was deployed and ready to use. Click “OK”.

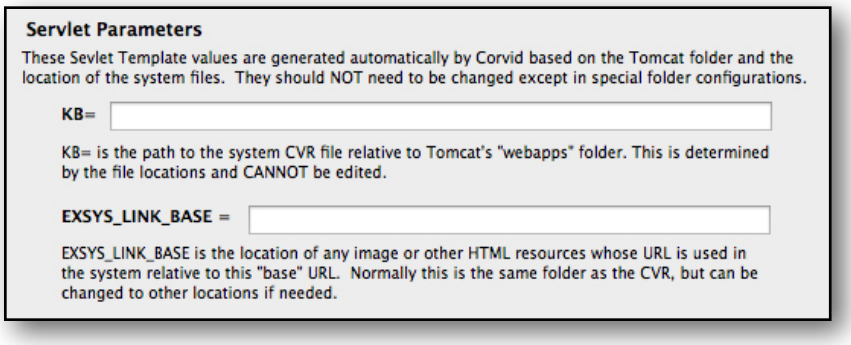


    If a copy of the Corvid Servlet has already been deployed, it will have created a folder named CorvidCore in the Tomcat webapps folder. To deploy the new copy of the servet, this folder must be deleted and rebuilt. Corvid will do this for you automatically, but **any files in the CorvidCore folder will be deleted**. Corvid will warn you about this. Because of this, the CorvidCore folder in webapps should **NOT** be used for your system files.

14. Make sure Tomcat is running. (If it is not click the “Start Tomcat” button) Click the “Test Corvid Servlet Runtime” button. This should display a URL in Safari saying “The Corvid Servlet Runtime is installed and running”. Close Safari. (If the “iInstalled and running” screen is not displayed, go to the Tomcat folder and open the “webapps” folder. Make sure “CorvidCore.war” was copied there and the folder “CorvidCore” was created. If not, Tomcat may have been put in a location that Corvid does not have permission to write to. In that case, you may need to move the Tomcat folder to a location that can be written, such as your User folder.



15. Do not put anything in the “KB=” or “EXSYS_LINK_BASE=” edit boxes. Corvid will fill these in automatically based on where your Corvid system is stored in the Tomcat folder.

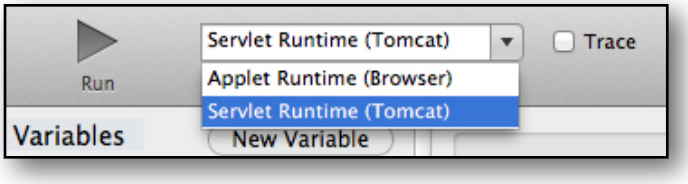


16. If you will not be using Tomcat immediately, you can click the “Stop Tomcat” button to shut Tomcat down. This will turn the green “Running” message back to the red “Not Running”message. Remember to start Tomcat back up before you use it to run a system.
17. Click the “Done” button on the Corvid Tomcat setup window. Whenever you start Corvid these Tomcat settings will automatically be loaded and used.
18. That’s all there is. You are now ready to build Corvid systems that run with the Corvid Servlet Runtime.

14.3 Running Systems with the Corvid Servlet Runtime

Once Tomcat is installed, it is just as easy to build systems for the servlet runtime as with the Applet Runtime. Most of the settings made for the user interface with the Applet Runtime are automatically converted by Corvid to CSS styles and applied in the same way via the servlets HTML user interface. This means that if you’ve already set fonts, colors, reports, etc in the applet mode, those will automatically convert over to comparable look-and-feel when using the Corvid Servlet Runtime.

To run with the Servlet Runtime just select “Servlet Runtime (Tomcat)” in the “Run With” drop down at the top left of the Corvid window.



If Tomcat is not running, or Corvid is not sure Tomcat is running, the Tomcat Setup window will be displayed. In that case, if the green “Tomcat Running” message is not displayed, click the “Start Tomcat” button and wait for the “Tomcat Running” message to be displayed. (Systems that start Tomcat slowly may require this to be done twice)

Servlet Runtime and Trace

The “Trace” option can be selected with the Corvid Servlet Runtime, but the trace information is not as extensive or interactive as using Trace with the Applet Runtime. The servlet will display the trace history at the end of each screen displayed. This is often enough to diagnose any problem, but complex logic issues may be easier to diagnose using the Applet Runtime. To do this, just switch the run mode back to the applet, use trace to diagnose the logic and then return to the servlet runtime.

The default servlet templates have a replaceable parameter CORVID_TRACE. This will be replaced by the trace information when a screen based on that template is displayed. The trace text is styled by a CSS style that can be modified in the templates. To use trace with the servlet runtime, any custom template files that are created should keep the CORVID_TRACE parameter in them.

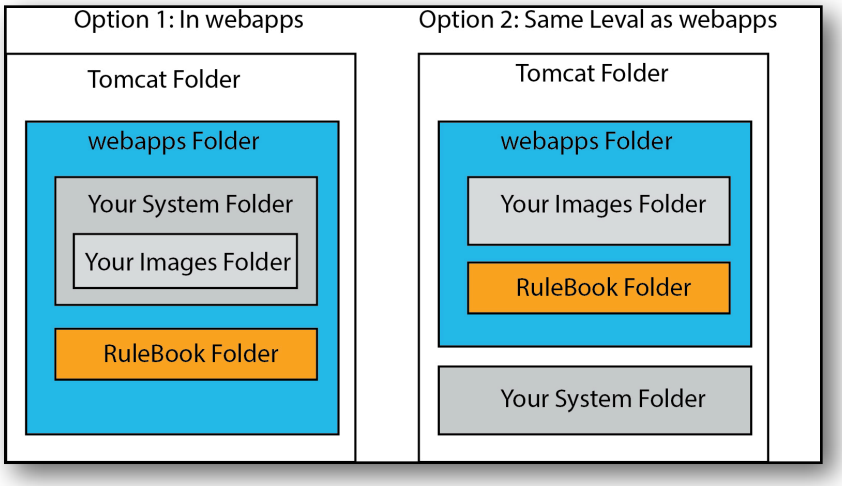
14.4 Where Systems Must be Stored

When building and running systems with the Applet Runtime, they can be stored anywhere on your Mac. However, the Corvid Servlet Runtime requires that they be stored in a particular place. The Corvid Servlet needs to be able to access the files in the Tomcat environment, so **they MUST be under the Tomcat "webapps" folder** or a folder at the same level as "webapps".

Note: Systems should NOT be put in the "CorvidCore" folder in webapps.

The "CorvidCore" folder is automatically created by Tomcat when the Corvid servlet (CorvidCore.war) file is deployed. This rebuilds the folder and anything added in that folder will be lost.

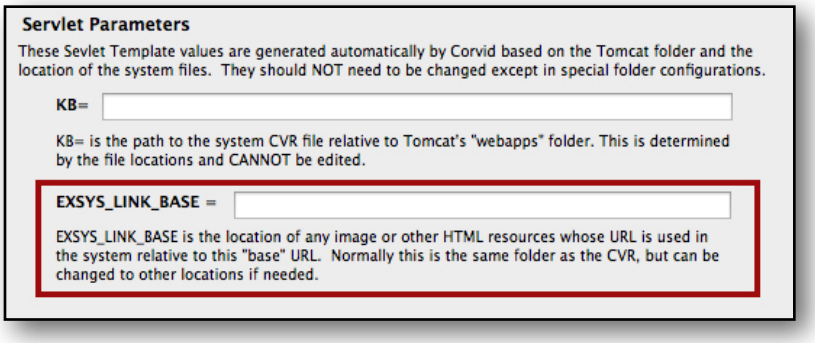
If using Option 1, your system folder should be at the same level as "CorvidCore", but not in that folder.



When first starting a new Corvid system it is easiest to put the systems in Tomcats webapps folder. Just create a folder in Tomcat's "webapps" for your Corvid systems. It is generally best to create a folder for each Corvid system to keep all the associated files together. This folder can have subfolders for images or other needed files, or these can just all be put in the same folder. Organizing each system in its own folder will make moving the system to a production server easier later, and will simplify customizing template files. In this case, Corvid will automatically handle all the links to images.

Putting the System Files at the Same Level as webapps

An alternative is to put your system folder at the same level as webapps, rather than in webapps. This has intrinsically higher security, but requires that any images or other files referenced by a URL still be put in their own folder in webapps, or on another server. In this case, the EXSYS_LINK_BASE option in the Tomcat Setup screen needs to be set to the URL of the folder with the images.



When a file is outside of webapps, the contents of the file is not reachable from a browser by a URL. This is what gives this approach much higher security, but it also means that any image file (or other file referenced by a URL) can also not be reached. Consequently, images and anything else that is referenced by a URL from the system MUST be placed in a folder that **can** be reached. The easiest approach is to put these in folder in a folder in webapps. They can alternatively be put on another server anywhere on the web, however, the Corvid system needs to know where they are located.

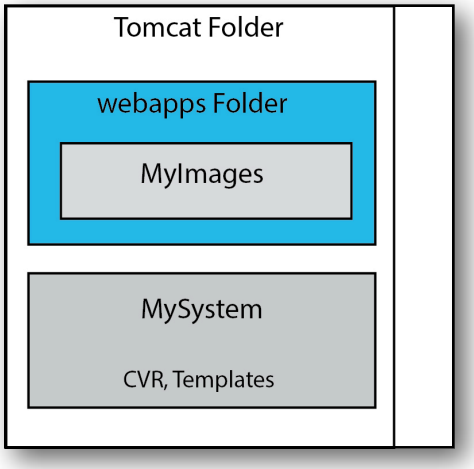
To do this, open the Tomcat Setup window for the system and set the EXSYS_LINK_BASE to the full URL path for the **folder** with the images. All images need to be in the same folder or a subfolder.

For example, for a system in the folder MySystem with the system files .CVR, and possibly template files, with the images in the webapps folder, in a folder named MyImages.

The EXSYS_LINK_BASE value would be set to the full URL of the folder with the images. In the default local Tomcat, this would be:

http://localhost:8080/MyImages/

(**Note:** the EXSYS_LINK_BASE string ends in a /. The image filenames will be appended to this to make the actual URL for the image)



Security Issues When Fielding Systems

In development, the system is just on the local Mac, so external access is not an issue. However, when systems are fielded on production servers, there are more concerns. Files the webapps folder may have security issues since the actual system files can be accessed externally with a web browser.

If there are security concerns about access to the system files, it is best to use the approach of having the system folder at the same level as webapps (NOT in webapps).

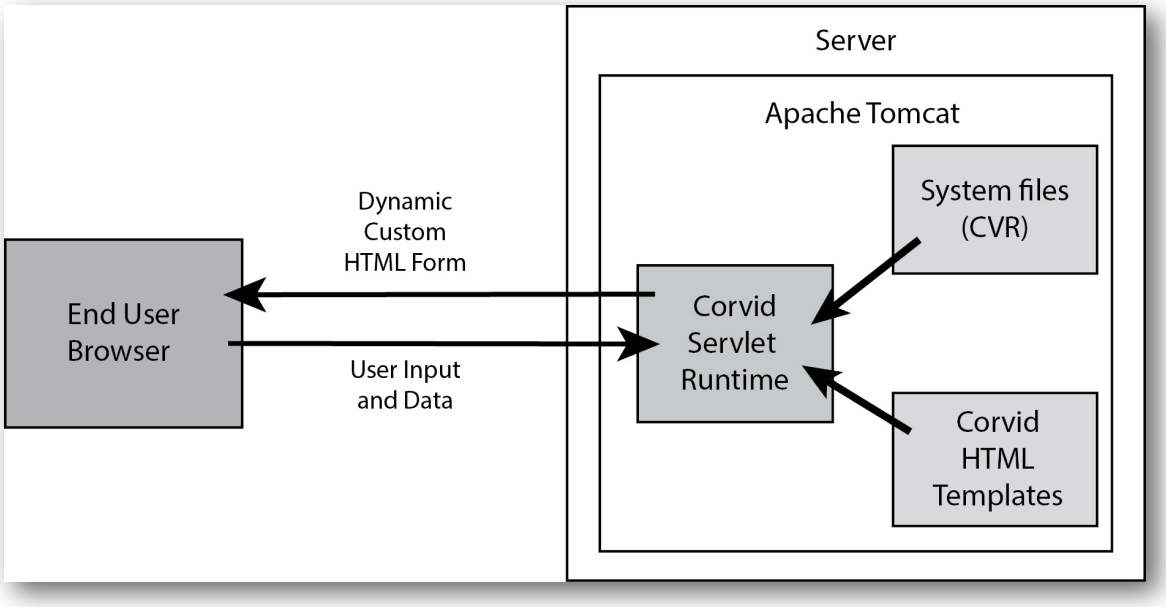
An alternative is to put the system folder in webapps, but modify the WEB-INF file for Tomcat to block access to files of concern - however, this requires advanced knowledge to Tomcat administration. If you have access to IT staff familiar with Tomcat administration, this may be an option for you.

- If a system only uses text and does not use images or linked HTML pages, use Option 2 (same level as Tomcat). It will be secure and images will not be a problem
- If a system is not sensitive, and there is no concern about external access to system files, use Option 1 (system files in webapps). Corvid will take care of all the links automatically.
- If a system is sensitive or there is concern about external access to system files, use option 2 (same level as webapps). This will require that EXSYS_LINK_BASE be set in the Tomcat setup window to indicate where images, etc are stored. It is also possible to use option 1 (in webapps) but modify the Tomcat WEB-INF data to not allow certain files to be served. This requires an advanced knowledge of Tomcat administration. If you want to go this way, check with your IT staff for assistance.

If you are just starting with the Corvid Servlet Runtime, use Option 1 for your first systems. It makes development easier and a system can later be moved to Option 2, with only a little editing.

14.5 Templates and the End User Interface

The Corvid Servlet Runtime runs on the server under Tomcat (or equivalent program) and processes the Corvid system logic the same as the applet. It asks the same questions for the same reasons and comes to the same conclusions and recommendations. The difference is in HOW it interacts with the end user.



With the Applet Runtime, the end user opens a single HTML page that has an applet window in it. The system runs in that applet window. When the system needs to ask a question, it just displays the question in the applet window in the HTML page that is already displayed. As more questions are needed, or results need to be displayed, the content of the applet window keep changing. This is because the Corvid system is actually running on the end users machine and controlling the content of the applet window.

With the Corvid Servlet Runtime, new HTML pages are dynamically generated each time the system needs to ask a question or display a result. Each of the pages is an HTML form that the end user can interact with to answer questions and provide data or just click an OK button. It is up to the Corvid Servlet Runtime to dynamically build each of these HTML forms as they are needed. Since a system may need to ask may questions based on the system logic, there may be many, many different possible HTML pages that could be needed to run the system. The Corvid Servlet Runtime handles this potential complexity by using generic HTML templates that can be used to build a wide variety of question or result screens.

The templates combine standard HTML code with special Corvid replaceable parameters that are replaced by the content appropriate for a specific screen. This allows a single template to ask a wide variety of questions with a standard look-and-feel. For example, the replaceable parameter VARIABLE_PROMPT will be replaced by the prompt for whatever variable is currently being asked. Since VARIABLE_PROMPT appears in the HTML template as standard text, any HTML layout or CSS style that is applied to VARIABLE_PROMPT will apply to the actual prompt text when it is displayed in the dynamic HTML page.

Any Corvid system can be run with the 3 default templates the come with Corvid - one to ask questions, one to display results, titles and reports and one (rarely needed) one in case the end of the command block is passes.

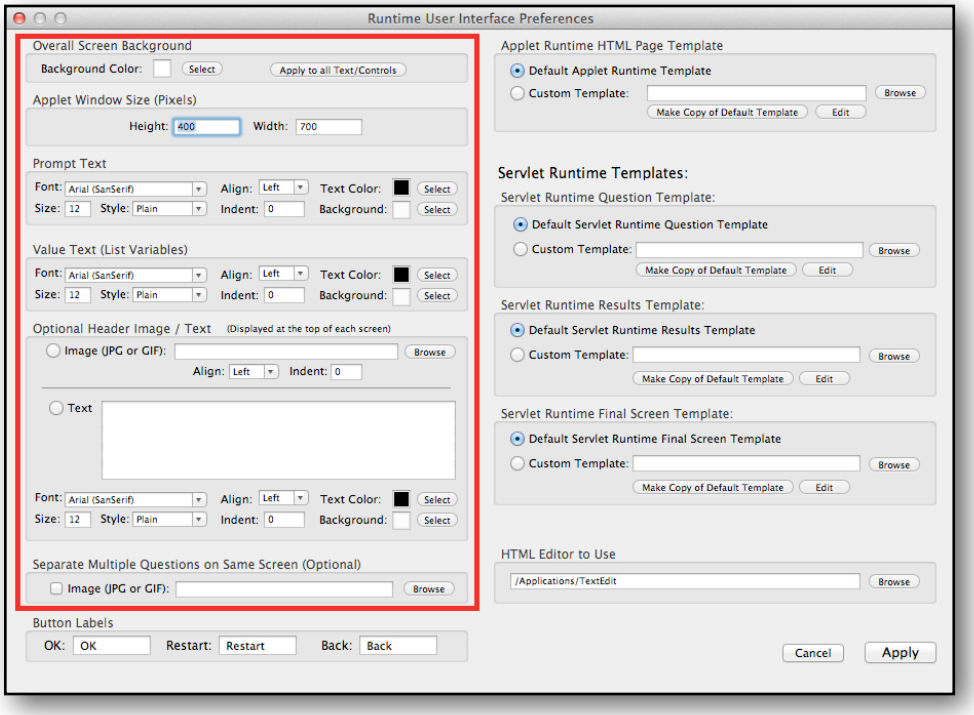
The default templates use the same fonts, colors, alignments, etc set in the User Interface window when running with the Applet Runtime. In the servlet runtime, these are converted to CSS styles and HTML commands internally and incorporated into the pages dynamically built from the template. The content (text, images, variables, etc) that are selected for a results screen automatically get converted to HTML and added to the results template.

This makes it very easy to change the appearance of a system when using the Servlet Runtime even if you don't know HTML or CSS, Corvid handles everything for you. For more complex interfaces, it is easy to copy and edit the default templates to add a look that goes beyond what can be done from within Corvid. Anything that can be designed or done in HTML (including HTML5, CSS3, JavaScript, AJAX, etc) can be added when it is needed. "Generic" templates can be applied to the entire system, or individual templates can be associated with individual variables, questions and result screens.

14.6 The Easy Way - Running with the Corvid Servlet Runtime Defaults

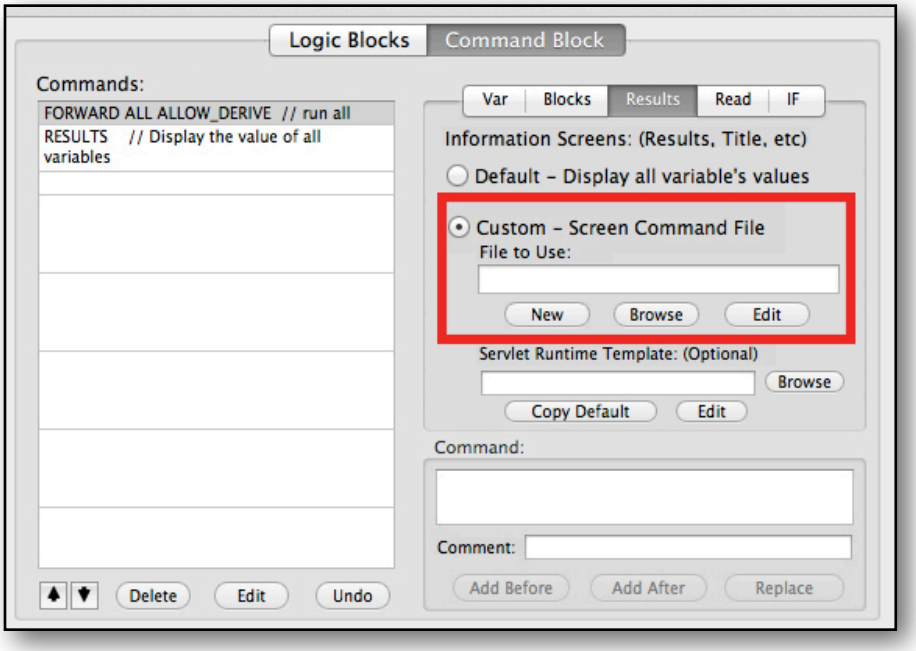
The Corvid Servlet Runtime comes with default templates. These automatically incorporate the same general look and feel as when running with the Applet Runtime and are set the same way from the User Interface and Result command windows. Corvid handles everything automatically with the default templates.

Just build the system logic as with the Applet Runtime. To modify the way questions are asked, open the User Interface window.



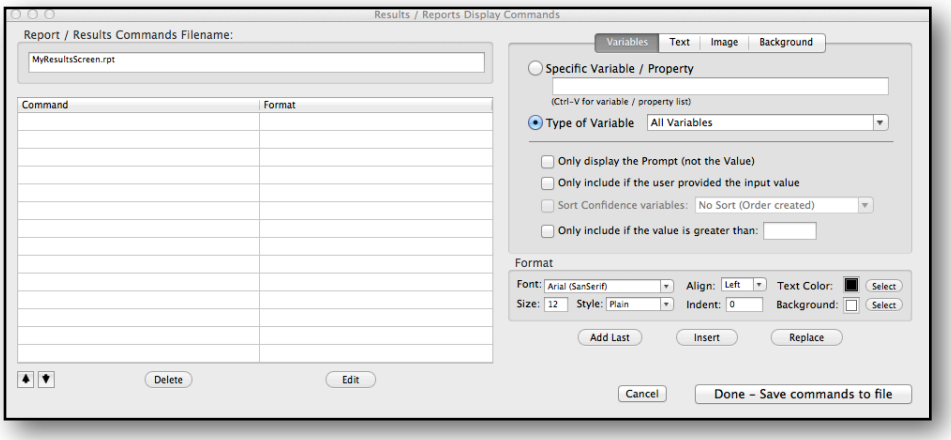
Any of the commands marked in red above also apply to the Corvid Servlet Runtime default templates. The User Interface options are converted to CSS styles or HTML code embedded in the default servlet template. This makes it easy to change the look and feel of a system without having to modify the HTML.

The default servlet Results template will incorporate the commands added to any DISPLAY command. In the Command Block section, go to the “Results” tab and select the “Custom - Screen Command File” .

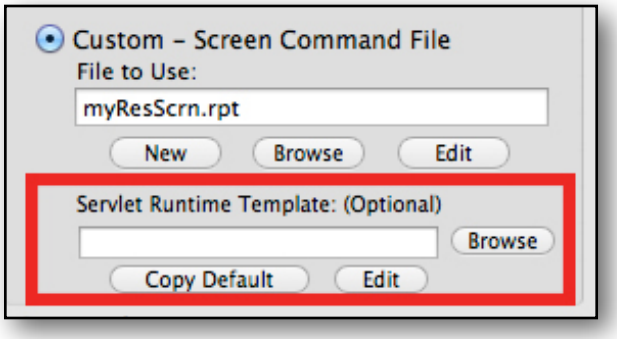


Add any commands just as when building a custom results file when using the Applet Runtime. The results commands can include variables, text (including text with double square bracket embedded variables), images, background colors to build up a report or result screen. The font, size, color, alignment, etc for each item can be set at the bottom of the window.

Corvid will automatically convert these results commands into CSS styles and HTML commands and embed them in the default Results screen.



To have the default screen used, **leave the edit box for specifying the “Servlet Runtime Template” blank**. This edit box is only for using a customized Results screen. (This is covered in the next section). However, it is necessary to specify the Custom Screen Command File that holds the screen commands. This is done exactly as with the Applet Runtime.



Once the User Interface options and Result commands are set, just select to run with the Corvid Servlet Runtime and Corvid will take care of everything automatically.

By using the options in the User Interface window and Custom results screen commands, it allows very nice looking systems to be created, but Corvid allows you to create much more complex and interesting user interfaces by copying and modifying the HTML template files. This requires knowledge of HTML but since it is done using templates, usually only 2 customized templates are all that is needed to completely change the look of a system.

14.7 Customizing the Templates - Creating Complex User Interfaces

Corvid includes 3 default templates that are used by the Corvid Servlet Runtime:

- **Question Template** - use to ask the end user for input.
- **Results Template** - used for title screens and anywhere the RESULTS or DISPLAY commands are used.
- **Final Screen Template** - displayed if the end of the Command Block is reached. (Generally not needed if the Results template have been modified).

These 3 templates can be modified to become the new system defaults, or multiple different templates can be used for individual questions (variables) or result screens.

This requires some knowledge of HTML. The details are covered in chapter 16.

14.8 Moving Systems To a Production Server

In most cases, moving a finished system to a production server running Tomcat is very easy and files are just put in the same folder arrangement as the copy of Tomcat on the local Mac.

1. Copy the file CorvidCore.war to the webapps folder of Tomcat on the production server. This should deploy and create a folder named CorvidCore.
2. Move the folder with your system files to the production server in the same place in the Tomcat folder structure.
3. Run you system in the **development** environment and copy the URL for the first screen. This will be something like:

   http://localhost:8080/CorvidCore/CorvidSR?KBNAME=../MyDir/MySystem.CVR

   Open a browser window and paste the URL, but change the “localhost:8080” to the address for your server (e.g. http://www.exsys.com/CorvidCore/CorvidSR?KBNAME=....) This will run the system from your server.
4. If it was necessary to put the system in a different folder structure on the sever, the KBNAME= parameter will need to be modified. It is the path to the CVR file relative to the servlet running in the CorvidCore folder in webapps. So the ../ drops down a folder to webapps, then the path moves up the folder(s) to where the CVR file is located. If you put the system at the same level as webapps (rather than in webapps) for security reasons, use ../../ to move to the folder that holds webapps and move up into the folder with the CVR.

5. If your system uses EXSYS_LINK_BASE=, that may need to be modified for the structure on the production server and the CVR file rebuilt. Not doing this can result in a system that runs correctly on your machine (which can reach a local folder specified by EXSYS_LINK_BASE=), but but will not run correctly for other users.
6. Test run the system. The most common error is to not have all the ancillary files (images, etc) in the correct location. If you see icons in the browser for missing images, examine the HTML page source and find the problem reference. Then add/move the image to the correct location or change the EXSYS_LINK_BASE=.

Systems can also be moved to production servers running Glassfish, IBM Websphere and other “servlet containers”. Check with your system administrator for assistance in fielding servlets in these environments.

15 Collection Variables and Reports

15.1 Collection Variables

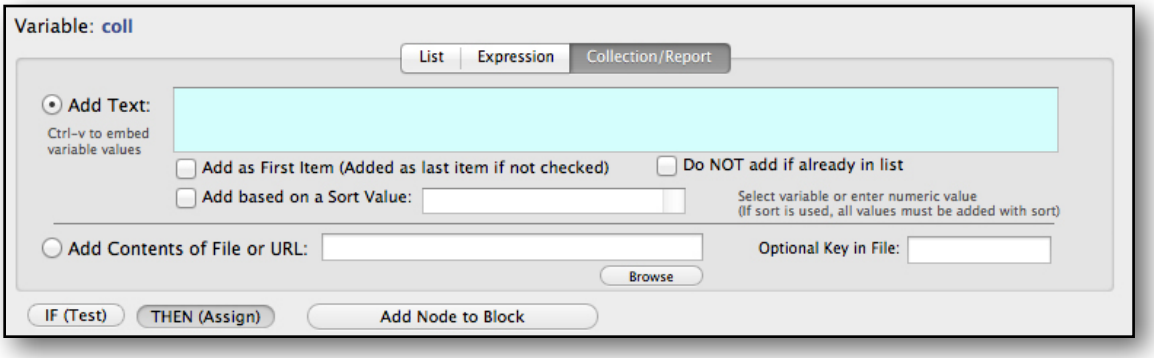
Collection variables are a special type of Corvid variable. They are very useful for many advanced systems and are especially well suited to building reports.

The value of a Collection variable is a list of strings. One way to think of it is like a shopping list and Corvid commands can add items to list or check to see what is in the list. However the items in the list can be much more complex than just a list of groceries. The strings can be HTML code, complex text or anything else needed to build a report. Items can be added to the list by a series of commands, or read from a file.

A file can be created as text or HTML with Corvid variable names or properties included in double square brackets. If this file is read into a Collection variable, Corvid will replace the double square bracket variables with their current value to build a report in the Collection variable that can then be displayed.

15.2 Adding Values to a Collection

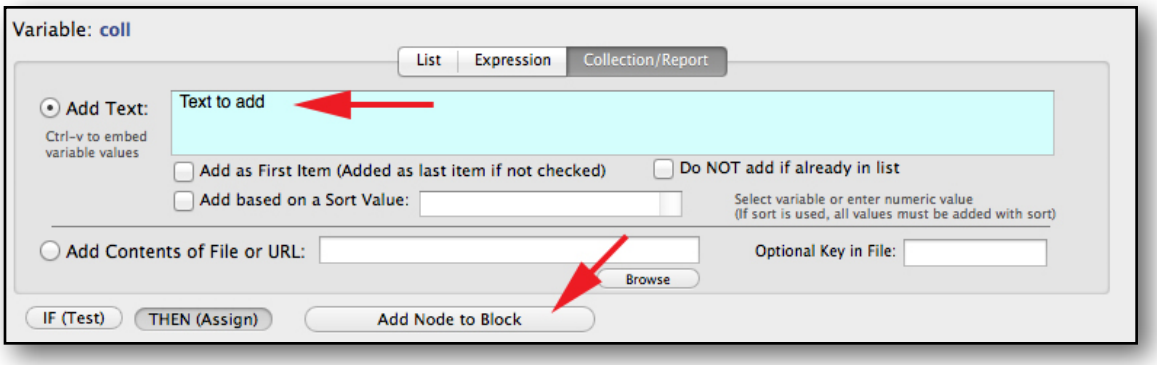
Whenever a Collection variable is selected in the Variables list to build a THEN node in a Logic Block, the node builder will change to the options for adding to the collection.



This panel makes it easy to build the various commands to add content to the Collection variable. There are 2 main types of commands - adding text that is entered in the blue edit box and adding text from a file.

Add Text

To add text, enter the text in the blue edit box and click the “Add Node to Block” button.



This will add a node to the Logic Block with the ADD method for the Collection variable.


```
[varname.ADD] text
```

Notice that the text to add is outside of the closing square bracket.

The text added to the collection can include other Corvid variables embedded in double square brackets. This is a very powerful technique since it allows adding the value of variables along with putting them in a text context useful in a report. If the value of an embedded variable is not already set or derived, Corvid will immediately invoke backward chaining to derive the value or, if it cannot be derived, asked of the end user. For example, the text to add to the collection could be:

```
Name: [[name.value]]
```

When this is added to the collection, the double square bracket embedded variable [[name.value]] will be replaced by the value of the variable [name]. If [name] does not already have a value, Corvid will try to derive one from the rules using backward chaining. If that cannot set a value, the end user will be asked for the value. This is very useful since simply adding the embedded variable will cause it to be derived / asked when it is needed without any other procedural commands to force this to happen.

If the text will be used in an HTML display, it can also include HTML tags. For example, the text to add could be:

```
<span class="NameStyle"> Name: [[name.value]] </span><br />
```

This uses CSS to style the text when it is displayed using a NameStyle from CSS that will be added later.

A single text string added can include multiple embedded variables and be as long as required.

To add the current value of a variable, without it being derived or asked, precede the variable name with an asterisk:

```
[[*varname]] or [[*varname.property]]
```

Each of the various commands to add to the collection have different “method” names that can be recognized in the Logic Block. These are controlled by the options under the blue edit box.

The new item will automatically be added as the last item in the list. If the item should instead be added at the top of the list, check the “Add as First Item” check box. This will build a node with the ADDFIRST method: [varname.ADDFIRST] text

In some systems, multiple rules can each add the same item to the list, but you may not ever wanted added more than once. This can be achieved by checking the “Do NOT add if already in list” check box. This will build an ADD_UNIQUE (or ADDFIRST_UNIQUE) method that will first check the list and only add the new item if it is not found.

For example, a system to select what to take camping might have rules:

IF

The season is winter

THEN

[clothing.ADD_UNIQUE] hat

IF

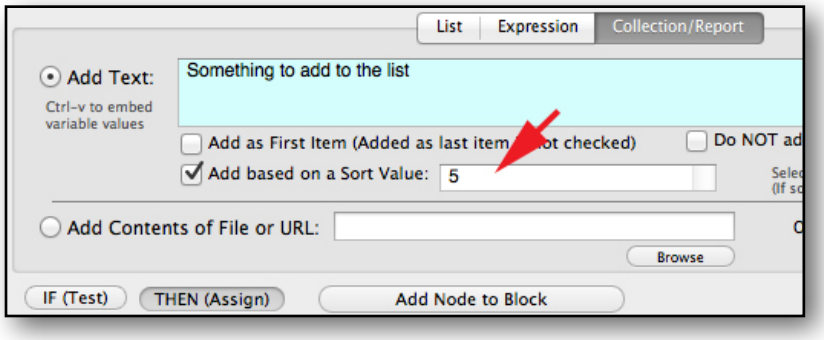
Weather forecast is rain

THEN

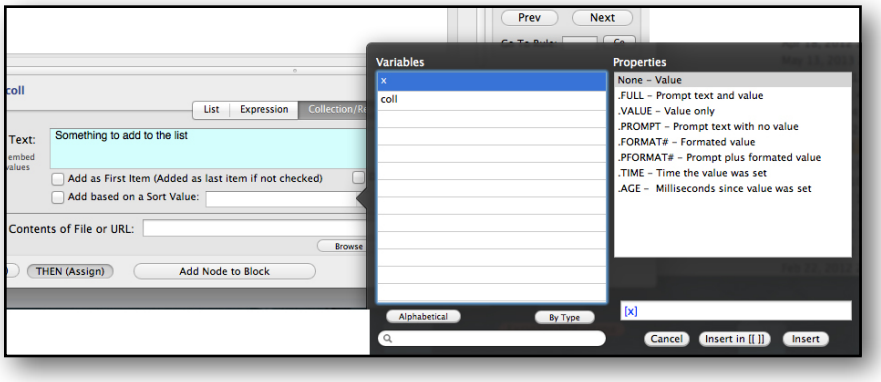
[clothing.ADD_UNIQUE] hat

These rules will remind the user to bring a hat if it is winter or if rain is forecast, but will not add it to the list more than once. The rules in the system can build up a list of cloths to bring and display it in a report at the end.

Add Sorted



A special option allows items to be added to the list based on a sort value. This is used in place of the ADD or ADDFIRST option to insert items into the list based on a numeric value. The item with the highest value will be the first item in the list. The item with the lowest value will be the last one in the list. **If the sort option is used for a Collection variable - ALL of the items must be added to that collection using sort.** Do not mix the normal ADD, ADDFIRST or UNIQUE options with ADDSORTED.



The sort value can be a simple numeric value entered in the sort edit box, or it can be an legal Corvid expression that evaluates to a numeric value. To add variables or expression to the edit box, use Control-V to display the variable list popup.

Creating a sorted list can be very useful in advanced systems when the list needs to contain many items, but there is a significance to where in the list items are displayed. Systems can calculate a ranking of "how good" an item is for the end user and add it to a sorted list based on that ranking value. If this is done for multiple items, the ones with the highest score will "float" to the top of the list and can be displayed to the end user as the "best" choices.

15.3 Add From a File

The most powerful and useful way to add content to a Collection variable is to add the contents of a file. This adds lots of content in one command, and any double square bracket embedded Corvid variables will be replaced by their value, make it easy to build report templates in HTML.

To add the contents of a file or URL, either:

- Enter the name of a file that is in the same folder as the other system file (or in a subfolder with a path). This can be done with the Browse button.
- Enter a URL to a file anywhere on the web. This can include anything that can be referenced by a URL so Java servlets or other active content can be used.

This will build an ADDFILE command that will add the contents of the file. Each line in the file will become a separate item in the Collection variable's value list.

For example, if a text file was created named pets.txt that had 3 lines:

dog

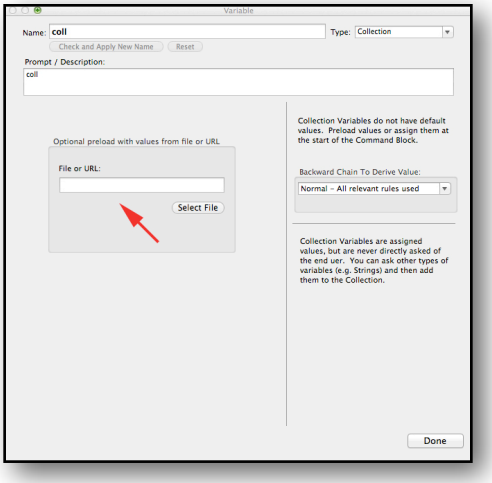
cat

bird

The command `[coll.ADDFILE] pets.txt` would add the 3 items to the Collection variable coll.

Initialization

Besides using a command to add the contents of a file to a collection, a collection can be initialized with the contents of a file. This fill will be read at the start of a session and each line will be added to the collection. Add the file to read when the Collection variable is created or edited. This can be useful for adding "header" content to a collection which will then have other content added by the system rules.



The file read in should **not** contains double square bracket embedded variables. If it does, these will be asked/derived, but since this is being read during initialization, not all variables may have yet been initialized and can result in unexpected behavior. If the file has embedded values, it is better to read it as the first command in the command block which will happen after all variables have been initialized.

Optional Key Identifiers in Files

It is possible to read only a section of a file into a Collection variable by using key identifiers to select a potion of a file. This is done by entering a key string in the “Optional Key in File” edit box when building the ADDFILE command. Then in the file, select the associated text to add by adding:

```
<!--  Corvid_KEY=key_str  -->
```

text to add

```
<!--  Corvid_KEY_END=key_str  -->
```

If there is a key string, Corvid will read down the file until it finds the line

```
<!--  Corvid_KEY=key_str  -->
```

matching the key string. It will then add all lines in the file down to the closing:

```
<!--  Corvid_KEY_END=key_str  -->
```

Make sure that each Corvid_Key has a matching Corvid_key_end for the same key string.

A file can have multiple overlapping Corvid_key regions and all lines down to the closing Corvid_Key_End will be added, even if there are other Corvid_key markers in it.

Key strings can include spaces, are not case sensitive and leading and trailing spaces are ignored.

Key strings can be double square bracket embedded variables. A variable in single square brackets will not work as a key, but one in double square brackets will. This allows the keyed section of the file to read, to be set by system logic, usually by assigning a value to a string variable.

The Corvid_key markers in the file are HTML comments. This allows them to be added to an HTML file invisibly and to have markers included in the collection without them being displayed as HTML.

For example, if the pet.txt file had:

```
<!--  Corvid_KEY=dogs  -->
beagle
labradore
poodle
<!--  Corvid_KEY_END=dogs  -->
<!--  Corvid_KEY=birds  -->
parrot
conure
cockatoo
<!--  Corvid_KEY_END=birds  -->
```

Using the “birds” keyword, the command [coll.ADDFILE] pets.txt, birds would add only:

parrot
conure
cockatoo

If the pets.txt file was:

```
<!--  Corvid_KEY=dogs  -->
<!--  Corvid_KEY=All pets  -->
beagle
labradore
poodle
<!--  Corvid_KEY_END=dogs  -->
<!--  Corvid_KEY=birds  -->
parrot
conure
cockatoo
<!--  Corvid_KEY_END=All pets -->
<!--  Corvid_KEY_END=birds  -->
```

The “All pets” keyword would go across the other keys to set the collection to:

```
beagle
labradore
poodle
<!--  Corvid_KEY_END=dogs  -->
<!--  Corvid_KEY=birds  -->
parrot
conure
cockatoo
```

This would include the 2 extra key lines, but if this was being used in HTML, these would be invisible since they are HTML comments.

Keys can also be used to read self-contained blocks of HTML code into Collection variable so that it can be pasted into other HTML templates. This is a block of HTML that starts with a tag and ends with the closing tag, though it can have other correctly matched tags in it. For example:

```
<span>  .....  </span>
```

Just build a HTML page in an HTML editor that includes the block you want to add put in the Collection variable. Then wrap that section with Corvid_KEY markers:

```
<!-- Corvid_KEY=UsefulPart -->
<span> .....  </span>
<!-- Corvid_KEY_END=UsefulPart -->
```

and use the key when setting the collection. This allows the content to be edited with an HTML editor as a full page, but still be referenced in pieces. A single HTML page may have multiple sections with different keys.

Conditional Inclusion of Text

In addition to the "key" markers that limit the ADDFILE to a part of a file, sections of the file can be included or excluded based on boolean test expressions.

The syntax for conditional inclusion of text is:

```
#Corvid_IF expression
    text to include
#Corvid_ENDIF
```

NOTE: The #CORVID_IF commands must be on their own line in the file and should be the only text on the line.

The text to include can be any length and any number of lines.

The test expression is any Corvid expression that evaluates to TRUE or FALSE, such as:

```
([X] > 0)
[NAMES.INCLUDES Bob]
[X] > [Z]+5
```

The expression can include any Corvid variables and properties.

Inclusion tests can be nested. Each #Corvid_IF must have a matching #Corvid_ENDIF. The #Corvid_ENDIF corresponds to the first preceding Corvid_IF that does not have a matching Corvid_ENDIF. If a block of text contains other #Corvid_IF tests, the block included/excluded will be all of the text to the point of the #Corvid_ENDIF that matches the initial #Corvid_IF.

For example:

```
#Corvid_IF  test1
aaa
bbb
#Corvid_IF  test2
ccc
#Corvid_ENDIF
ddd
#Corvid_ENDIF
eee
```

If test1 is TRUE and test2 is TRUE, the lines included would be:

```
aaa
bbb
ccc
ddd
eee
```

If test1 is TRUE and test2 is FALSE, the lines included would be:

```
aaa
bbb
ddd
eee
```

Add	Add item to end of list
Add_Unique	Add item to end of list if not already in the list
AddFirst	Add item to top of list
AddFirst_Unique	Add item to top of list if not already in the list
AddSorted	Add with a sort value (text to add, sort value)
AddFile	Add contents of a file
DropFirst	Remove the top item in the list (No text needed in edit box)
DropLast	Remove the last item in the list (No text needed in edit box)
Remove	Remove an item from the list
Clear	Remove all items from the list

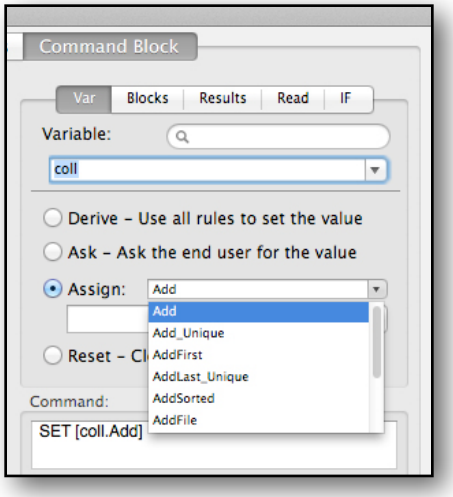
If test1 is FALSE, the entire block will be excluded, and test2 will never be tested. The lines included would be:

```
eee
```

The content to include must be designed so that various sections of text can be included or excluded and still produce valid syntax. This is especially important when using HTML.

One way to use this feature is to have a system that sets the value of various Confidence variables, then reads in a file that selects pieces of HTML content based on the variables that got high values. This allows a complex report to be build, while still keeping the Confidence variable text short.

15.4 Collections Assignments in Commands



When Collection variables are assigned a value in a command block using the SET command, there a method to use must also be selected. This is done from a drop down list that is active only for Collection variables.

Select the method to use and then enter the value in the lower edit box.

Besides the Collection variable methods used in Logic Blocks to add values to a collection, there are 4 special methods to remove items. These are used only in Command Blocks. The methods are

The Add methods are the same as in Logic Blocks. The DropFirst and DropLast methods do not need any text entered in the lower edit box. The “Remove” method can specify the item to remove by the text of the item, or by # followed by the number of the item to remove (e.g. [Coll.Remove] #3 will remove the third item in the list)

15.5 Content of Collection Variables

There are various properties to look at the content of a Collection variables.

Property	Value
No Property	The full text of the prompt followed by the values concatenated together with a space between them.
.FULL Separator	The full text of the prompt followed by the values concatenated together with the separator text between them.
.VALUE Separator	The values concatenated together with the separator text between them.
.COUNT	The number of values in the list.
.FIRST	The text of the first item in the list.
.LAST	The text of the last item in the list.
.ITEM #	The text of the item number # in the list.

Property	Value
.TOP #	The top # items in the list separated by a space.
.SCORE #	The sort value of item # in the list.
.CONCAT Separator	The values concatenated together with the separator text between them, but without padding.
.INCLUDES text	True if text is an item in the list, otherwise false.
.NOTINCL text	False if text is an item in the list, otherwise true.

These are covered in more detail in Appendix B.5.

15.6 Building Reports with Collection Variables

Corvid supports various types of reports to display the system conclusions and advice to the end user.

The simplest type of report is just the Results screen in the Corvid Applet Runtime. This allows displaying and formatting any of the variables in the system in the applet window that runs the system. Building this type of screen is covered in section 12.4 on the User Interface, and is built using the standard Corvid Screen Commands.

Any type of Corvid variable can be included in a report. The value for the variable may have come from user input or been set by system logic and rules. Collection variables are particularly useful since their value is free-form text that can be set by multiple rules that fire, or by reading from a “template” file with embedded variables.

There are several ways to build reports for the Corvid Servlet Runtime, depending on the control needed in the design of the report and the amount of HTML coding required.

Using the Default Servlet Templates to Display Reports

The easiest type of servlet based HTML report is to just let Corvid convert your applet screen design commands into HTML and display them. This is very easy to do.

Design your report with the applet screen commands described in sect 12.4.

This can include any types of variables, including Collection variables and can include both simple text where the format is set by the applet format commands (which will automatically be converted to CSS) and sections that include self-contained blocks of HTML. This means HTML that starts with an HTML tag and ends with the associated closing tag. There can be other HTML tags in the string, as long as they are closed or are single tags like
.

A good approach is to load a Collection variable(s) with HTML from a file. This can include embedded variables whose values will be replaced in the HTML. Then just display that collection(s) with the applet command defining the screen.

Move the system to a folder in the Tomcat webapps folder if it is not already there. Then just select to run with the Servlet Runtime.

Corvid will do all the work and display the report content in HTML.

Corvid does all the work, and the applet format commands will be converted to HTML, but it will be placed

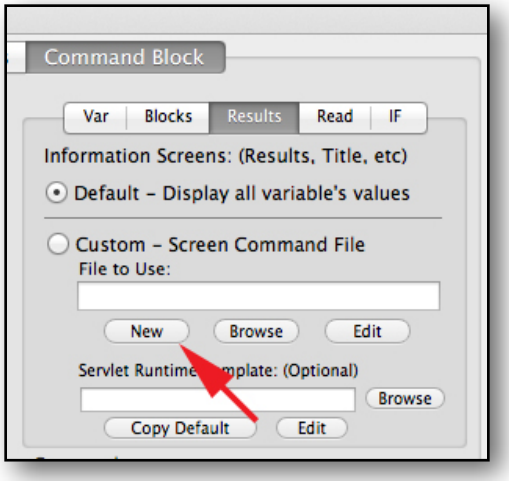
in the default Corvid results screen.

For example:

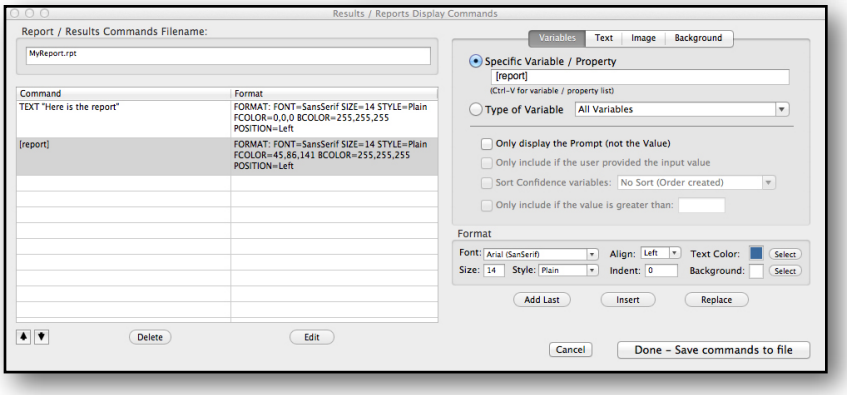
A system with a Collection variable named [report] could have content added by rules that add content, or which read content from a file:

```
IF
    ...
THEN
    [report.add]  A part of the report<br>

IF
    ...
THEN
    [report.add]  Another part of the report<br>

IF
    ...
THEN
    [report.addfile]  MyTemplateFile.html, contentPart
```

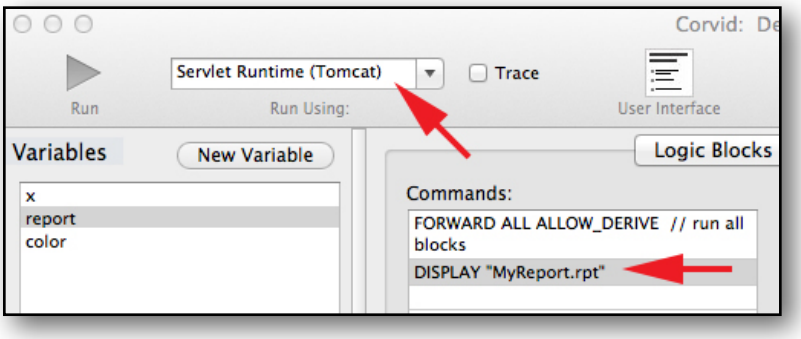
Then, using the applet screen commands, select to display [report]. Select to add a new Custom screen.



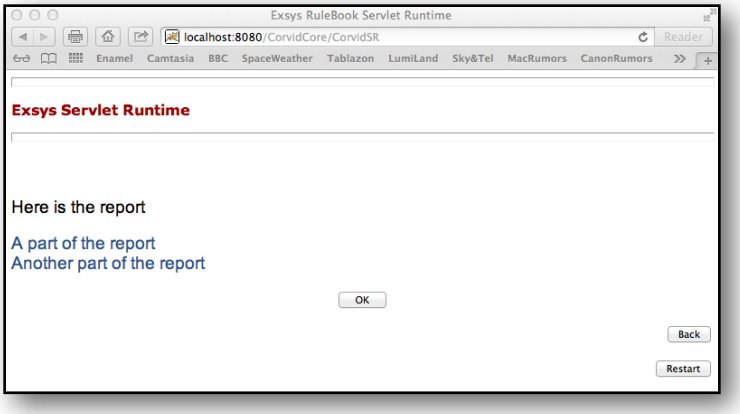
The custom screen can contain commands to display the report, along with text, images, other variables, etc. It can also use the Format options to set colors, fonts, etc.



Make sure the new custom DISPLAY command has been added to the command block and run with the servlet runtime.



Since there is no Servlet Runtime Template specific, Corvid will use the default template to display the results after asking any questions needed to run the system. It will use the commands and formats from the custom screen commands, but converted into CSS and HTML.



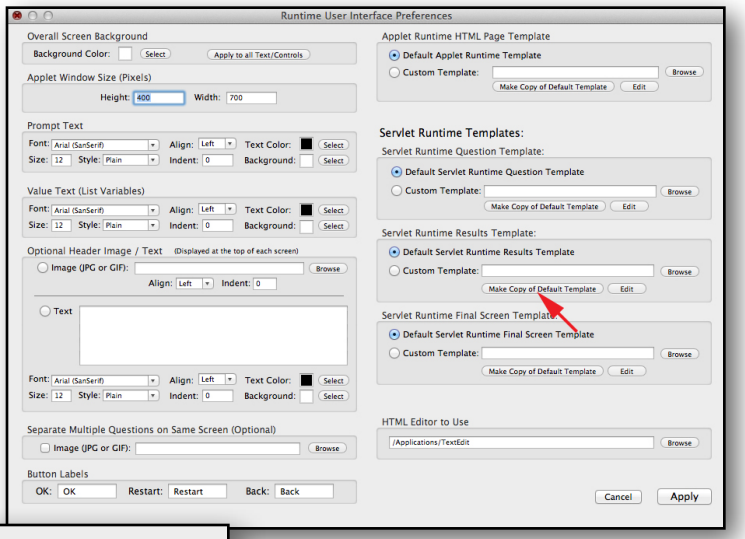
Customizing the Default Servlet Results Template

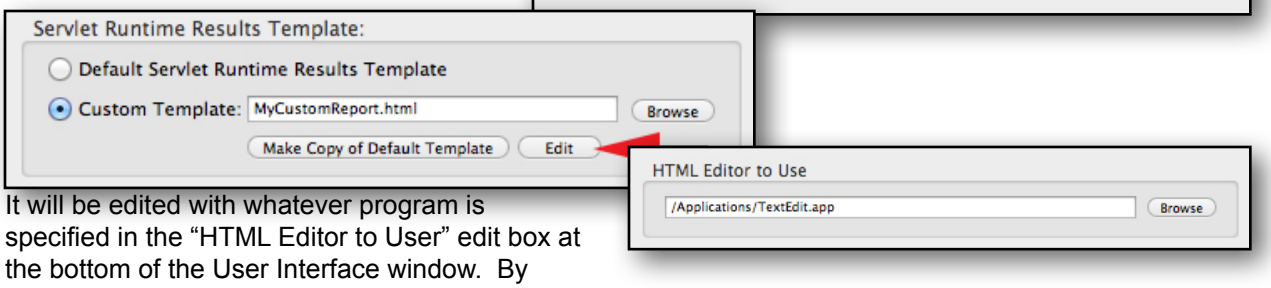
The above approach using the default Corvid Servlet Templates is easy, and does not require knowing any HTML. However, the applet commands are converted automatically and you may want to customize the report page to match your web site, add your own CSS styles, remove the “Exsys Servlet Runtime” at the top, etc.

This requires creating your own template to use. By far, the best way to do this is to start with the Corvid Default Servlet Template and edit it. This is easy to do.

Click on the User Interface icon at the top of the Corvid window to open the User Interface window.

In the Servlet Results template section, click the “Make Copy of Default Template” button. This will ask you to name the template. It is generally best to create this template in the same folder as your other system files. It will automatically be made an HTML file.

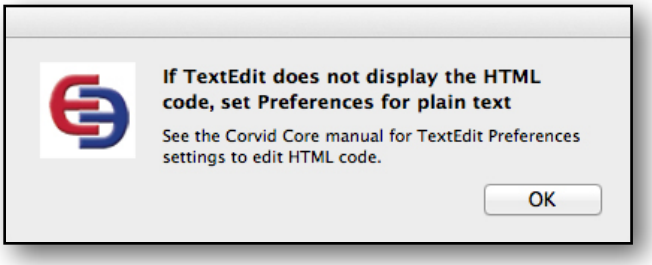




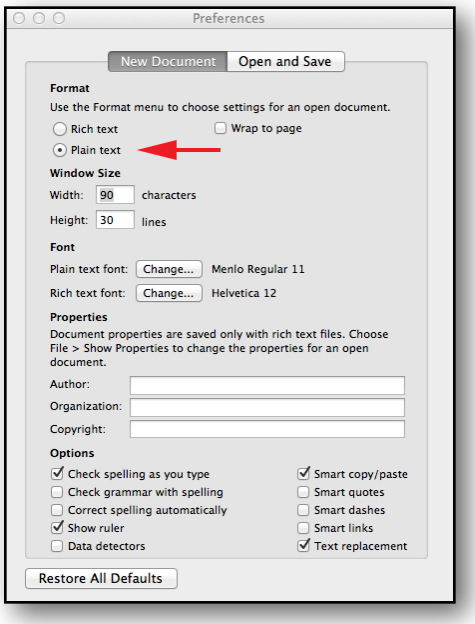
Once the new file has been created, click the “Edit” button to edit it.

It will be edited with whatever program is specified in the “HTML Editor to User” edit box at the bottom of the User Interface window. By default, this is set to TextEdit since it is part of the Mac operating system. However, it can changed to any other program that can edit HTML files such as Dreamweaver.

If it is left at TextEdit, you will get a warning message that TextEdit may not display the HTML code. This is because TextEdit can display HTML code as rendered HTML (like a browser window) or as the actual HTML code.



By default TextEdit is set to display the rendered version of HTML which cannot be directly edited. If this happens, open the TextEdit Preferences window and set it to “Plain text” and open the template again.



Customizing the Template Code

The actual HTML code for the template is a little complicated and there are parts you can safely change, some parts that should not be changed and some that can be changed, but it will reduce the functionality. It is important to make any changes in the correct sections. The details of editing the results template is in section 16.2.

16 Customizing Servlet Runtime Templates

16.1 Creating Complex User Interfaces

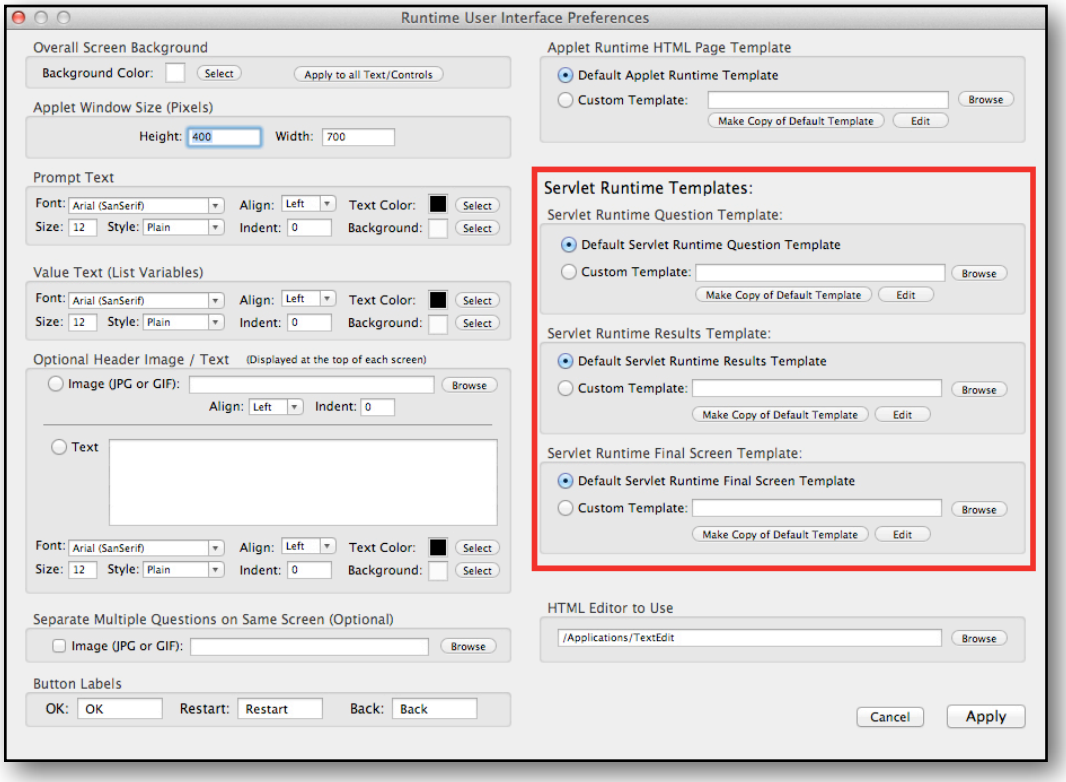
Corvid includes 3 default templates that are used by the Corvid Servlet Runtime:

- **Question Template** - use to ask the end user for input.
- **Results template** - used for title screens and anywhere the RESULTS or DISPLAY commands are used.
- **Final Screen Template** - displayed if the end of the Command Block is reached. (Generally not needed if the Results template have been modified)

These 3 templates can be modified to become the new system defaults, or multiple different templates can be used for individual questions (variables) or result screens.

This requires some knowledge of HTML.

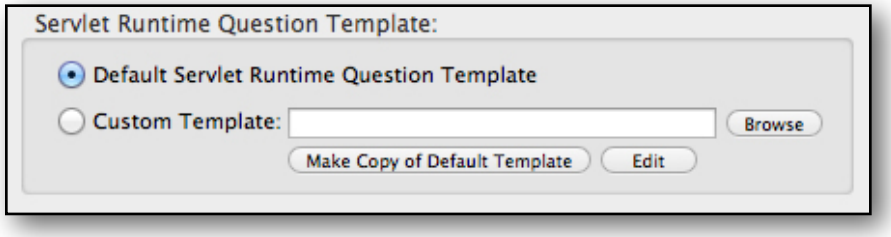
To change the templates open the User Interface window. On the right side there are options to change each of the 3 templates.



Each allows the original default template to be copied and renamed as a starting point. These can then be edited either outside of Corvid or with the HTML editor specified at the bottom right of the User Interface window.

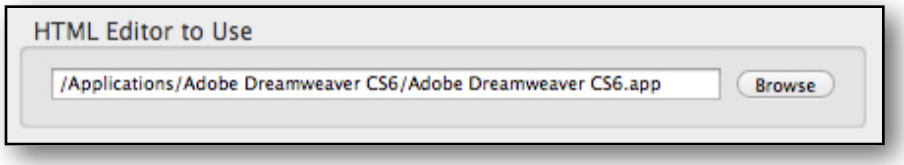
Each of the 3 default templates has a similar set of commands:

The default template can be copied and renamed by clicking the “Make Copy of Default Template”. Generally it is best to create the copy of the template in the same folder as the other files from the system that will use it.



An existing template can be selected by browsing to it.

Once a custom template is selected, it can be edited by clicking the “Edit” button. This will use whatever editor is specified at the bottom of the window in the “HTML Editor” section:



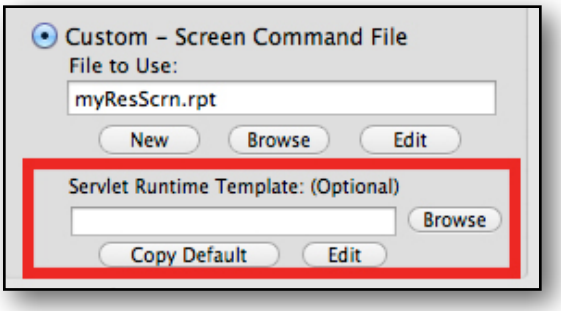
Customizing the Template Code

The actual HTML code for templates is a little complicated and there are parts you can safely change, some parts that should not be changed and some that can be changed, but it will reduce the functionality. It is important to make any changes in the correct sections.

If in doubt, don’t change something you are not sure of. Most items starting with EXSYS_ are replaceable parameters that the Corvid Runtime will replace by values set from within the Corvid Editor and passed in the CVR file. Some of these can be hard coded, but that will eliminate the ability to set them from within the Corvid editor, and the hard coded value will have to be made in all system template screens.

16.2 Editing DISPLAY Command Templates

DISPLAY commands are used to display results, titles or other information screens. Using them for reports is covered in chapter 15, but they can also be used for title screens and other informational screens. Just add a DISPLAY command at the appropriate place in the command block with an associated custom template.



Click “Copy Default” to make a copy of the default Corvid template for editing.

The same options apply when creating a custom system default Results template from the User Interface window.

Looking at the DISPLAY template code section by section:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />

<title>
Exsys Corvid Core Servlet Runtime
```

```
</title>
```

It is OK to change the title, or the charset if necessary

```
<script type="text/javascript">
<!--
function submit_form ( )
  {
    document.CorvidForm.submitButtonName.disabled = true;
    return true;
  }
// -->
</script>
```

Do not delete this JavaScript. It makes sure that end users do not click the submit button twice, which will cause a problem with the servlet

```
<style type="text/css">

.ExsysHeading {
    font-family: Verdana, Geneva, sans-serif;
    font-size: 16px;
```

Do NOT change the contents of the FORM tag. Corvid automatically inserts information here that is critical to having the system work correctly.

```
    font-weight: bolder;
    color: #900;
}

.TraceStyle {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 10px;
        font-weight: normal;
        color: #039;
        padding-left: 25px;
}

EXSYS_EXTRA_CSS

</style>
```

These CSS styles can be modified if needed. The ExsysHeadingCSS can be deleted if the section that uses it is also deleted. The TraceStyle only applies to the trace text and can be modified to make that text larger or displayed in some other way.the EXSYS_EXTRA_CSS replaceable parameters is where Corvid inserts the dynamic CSS styles built to handle the Format options. This should not be changed unless all options that can have Format commands in them (headers, footers, screen commands) are also deleted. Additional CSS styles that your code needs should be inserted here.

```
<BASE href="EXSYS_LINK_BASE">
```

The EXSYS_LINK_BASE replaceable parameter generally should not be modified. It can be changed to

a hard coded value, but then future changes will need to be made by editing the templates rather than from within the Corvid editor.

```
</head>


<body bgcolor="EXSYS_PAGE_BACKGROUND_COLOR"
```

Here too, the EXSYS_PAGE_BACKGROUND_COLOR replaceable parameter is set by the color selected in the Corvid editor and hard coding a value will eliminate that functionality.

```
onunload="document.CorvidForm.submitButtonName.disabled = false;">
```

Do not change this JavaScript. It is part of the code to prevent double clicks on the submit button.

```
<hr width="100%" size="10">
<p class="ExsysHeading">Exsys Servlet Runtime</p>
<hr width="100%" size="10">
<br><br>
```

This code can be changed to remove the Exsys header and to insert any HTML design for the top of the page.

```
EXSYS_HEADER
```

The EXSYS_HEADER replaceable parameter is replaced by the header image or text selected in the Corvid editor. This is the same header as appears on question screens. It can be deleted and replaced by hard coded HTML code, but this will eliminate the option to set it from within Corvid and should be done consistently in all question templates.

```
<form onsubmit="return submit_form()"   method="post"
action="CORVID_SERVLET" name="CorvidForm">
EXSYS_RESULTS_COMMANDS
```

The EXSYS_RESULTS_COMMANDS replaceable parameter is automatically replaced by HTML code built for any screen commands. If the template is used in conjunction with a file of screen commands, this should not be deleted. If instead, a more complex design is needed, this can be replaced by any HTML code with embedded Corvid variables to build and display the results screen. This allows any design desired to be implemented in HTML. If this is deleted, any screen command file associated with the RESULTS command will be ignored.

If the template is for other informational purposes, rather than displaying results, replace EXSYS_RESULTS_COMMANDS with any HTML to present the desired information.

```
<!-- Buttons_Normal -->
     <div align="center">
     <p><input type="submit" name="submitButtonName" value="   OK
"></p>
     </div>
     <div align="right">
     <p><input type="submit" name="~UNDO" value=" Back "></p>
     <p><input type="submit" name="~RESTART" value=" Restart "></
p>
     </div>
<!-- Buttons_Normal_End -->
```

```
<!-- Buttons_OneLine -->
     <div align="center">
         <input type="submit" name="submitButtonName" value="
OK   ">   
     <input type="submit" name="~UNDO" value=" Back
">   
     <input type="submit" name="~RESTART" value=" Restart
">   
     </div>
<!-- Buttons_OneLine_End -->

<!-- Buttons_Left -->
     <div align="left">
     <p><input type="submit" name="submitButtonName" value="   OK
"></p>
     <p><input type="submit" name="~UNDO" value=" Back "></p>
     <p><input type="submit" name="~RESTART" value=" Restart "></
p>
     </div>
<!-- Buttons_Left_End -->
```

The Buttons_ options and their headings allow the layout of the OK, Back and Restart buttons to be selected from within Corvid. There are 3 groups with different layouts. If you wish to change the button layout, delete the Buttons_OneLine and Buttons_Left groups, along with the comments <!-- Buttons_Normal --> and <!-- Buttons_Normal_End -->. Then edit the remaining button group for the new layout. The button labels (the value= parameter) can be changed, however the name= parameter should NOT be changed.

If a template will be the last screen displayed in a system, it is a good idea to delete the OK button and leave only the Restart and Back buttons. This prevents the end user from continuing past this screen and reaching the end of the command block which will result in a simple "System Done" screen.

```
</form>

<br><br>
<p class="TraceStyle">CORVID_TRACE</p>
```

The replaceable parameter CORVID_TRACE will be replaced by the trace text when running with trace turned on. This can be deleted for finished systems, but it will eliminate the ability for the template to run with trace. The format of the trace text can be modified by changing the TraceStyle CSS at the top of the page.

```
</body>
</html>
```

Customize as Much as Needed

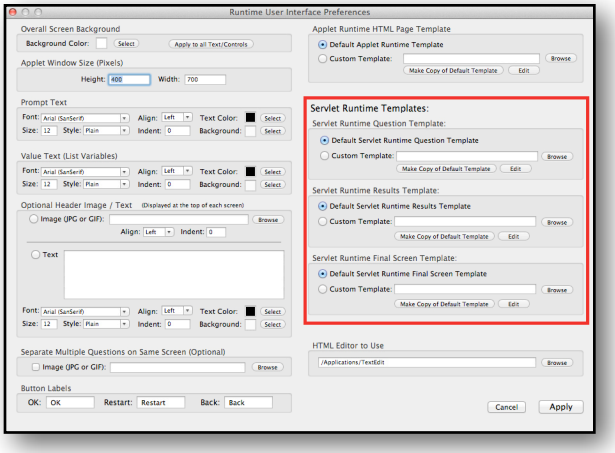
If you want to retain the ability to choose the variables to include in the results from within the Corvid Editor using screen commands, leave the EXSYS_RESULTS_COMMANDS parameter in and use your customized template in place of the default template. If you want a more complex design, delete the EXSYS_RESULTS_COMMANDS and instead add any HTML code using embedded Corvid variables in double square brackets.

One good approach is to use the EXSYS_RESULTS_COMMANDS during development. This makes it easy to change the information in the results to display more variables, trace, etc. Then when the system logic is finalized, replace the EXSYS_RESULTS_COMMANDS with a more complex hard coded HTML design for the information you want and delete the CORVID_TRACE.

16.2 Editing Question Templates

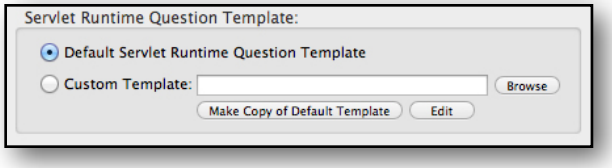
Systems will have one or more templates to ask questions. All systems will use the Corvid default question template to ask questions unless some other template(s) are specified.

The Corvid default question template can be customized to a system default template in the User Interface window.



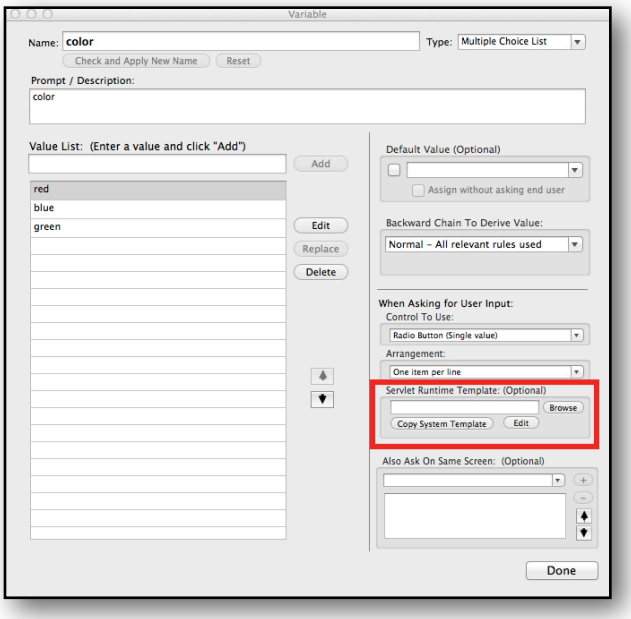
Select to Copy the Corvid Default Template, and then edit it.

When possible, it is good to have a single template apply to all questions in a system since that makes it



easier to keep a consistent look-and-feel for all questions. Question templates are designed to allow them to handle the different types of controls needed for different variables.

However, sometimes it is desired to have a separate question template for specific variables. Each variable can also have its own individual template specified to be used when that variable is asked of the end user. This is set from the edit window for the specific variable. (Select the variable in the Variables panel and click “Edit”). The controls allow a copy to be made of the system question template which can then be edited to build a template for the specific variable. If no template is specified for a variable, the system default question template will be used when (and if) the end user needs to be asked for the value of the variable.



How Question Templates Work

Question templates are probably the most complicated part of Corvid since they are designed to handle so many different options and variable types.

Question template files are a combination of HTML, special Corvid commands in HTML comments and Corvid replaceable parameters. The Corvid commands and replaceable parameters allow a template to be generic. This enables the single template to work for many variables, and on any server. A system using a very generic template often only requires only a single template for all questions, and it is easy to maintain. The extent to which a template is “generic” and designed using replaceable parameters determines how wide a range of variables it can be used for.

A template can be built without variable replaceable parameters, but it will be limited to a specific variable. Sometimes this is desirable, especially when using image maps or some other design that would apply only to a single variable.

A single generic templates that applies to all the questions in a system can be very effective and applies a consistent look-and-feel to all the questions in an easily maintained way. However, there is no reason not to have as many templates as needed to deliver a system with the desired end user interface.

Question Template Structure

Question templates are always an HTML form that will submit value(s) back to the Corvid Servlet Runtime. The question HTML form will always have:

- A <form> tag with an “action” to send the data back to the Corvid Servlet Runtime.
- One or more controls that allow the end user to input or select a value.
- A “Submit” button or action that will cause the data to be sent back.
- A way for the data sent back to include an identifier for the user’s Corvid session .

In addition to this, there can be any HTML code that the browser supports.

Corvid makes it easier to build these required items by using replaceable parameters and special Corvid commands.

The Corvid commands are added to the template as HTML comments - text between "<!--" and "-->".

This makes it easy to add them with HTML editors. Most of the commands mark a section of the HTML code that is only included in certain cases (e.g. only for certain variables or only if a Boolean test is true) or mark a block that is to be used repeatedly (e.g. repeated for each value in a List variable's value list).

The replaceable parameters are used to automatically assign text and values for the system. For example, a question template for a variable can use the replaceable parameter "VARIABLE_PROMPT". This will automatically be replaced with the actual prompt text for the variable. The "VARIABLE_PROMPT" string can be styled in the HTML page with CSS to set its style, color, size, etc. and that formatting will apply to the actual text of the prompt when it is replaced. If the prompt text is changed in the system, the template screen will automatically use the new text.

There are various replaceable parameters that can be used in templates to handle the prompts and values of variables and trace information.

The template files can have any name and extension, though .html and .tpt are two commonly used ones. Your templates are not required to have an .html extension but some HTML editors will only work correctly if you give the file a .html extension. A .tpt extension can be used with some editors and make it easier to recognize a template rather than a static .html page.

<FORM> Section

The form section of the template is used to ask the user a question using a variety of controls (radio buttons, edit boxes, etc). The user's input is sent back to the Corvid Servlet Runtime, which will process the data and continue the run. The HTML outside of the form section usually does not use Corvid commands and can be any HTML design.

There are 4 main sections to the form:

- The <form> tag. This specifies an HTML form and marks the start of the form section.
- The body of the form will have one or more controls that the user will use to provide input. This is usually done with one or more sections marked with “CORVID_ASK” commands that indicate sections of HTML code that should be used to ask certain variables.
- Within the CORVID_ASK section(s), there are usually Corvid replaceable parameters that will be replaced with the Prompt and Value, etc. from the variable.
- A “Submit” button(s) must be part of the form. This sends the data back to the Corvid Servlet Runtime.
- The closing </form> tag indicating the end of the form.

If you are familiar with HTML forms, the template can be built using a simple text editor such as Notepad, but it is generally easier to use an HTML editor.

<FORM> tag

The <FORM…> tag indicates the start of the form. All controls on the form must be between the opening <FORM> and closing </FORM> tags.

The typical form tag is:

```
<form  method="post"  action="CORVID_SERVLET" name="CorvidForm"
onsubmit="return  submit_form()">
```

Looking at the parts:

method="post"

The “method” portion of a <form> tag controls how much data can be sent back. In Corvid, all data should be sent back to the Corvid Servlet Runtime using “post”. This assures that any

amount of user input can be sent, even if large blocks of text are entered in an edit box, or there are many questions on a screen.

action="CORVID_SERVLET"

The action portion of a form tag tells the browser program where to send the data. This depends on the server location and the individual session. Corvid will automatically replace the "CORVID_SERVLET" with the correct information for the system and session. All you need to add is:

```
action="CORVID_SERVLET"
```

and Corvid will do the rest. If you look at an HTML page built from the template, you will see this converted to something complicated more like:

```
action="http://www.MyServer.com/CorvidCore/
CorvidSR;jsessionid=1583485A1EBB5"
```

Since Corvid automatically replaces "CORVID_SERVLET" with the servlet location and session information, a system can be run from any server without changing the settings,

name="CorvidForm"

The "name" is optional, but used in the JavaScript associated with the form. The name can be anything, provided it is used consistently in the JavaScript. To just use the standard Corvid JavaScript, keep the name as "CorvidForm"

onsubmit="return submit_form()"

The "onsubmit" is optional, but highly recommended. It calls a provided JavaScript function on the page that blocks the end user from inadvertently sending the same data multiple times, which will lead to the Corvid Servlet Runtime reporting an error. This is easily fixed with the standard Corvid JavaScript to add to the page. This also requires that the form be named "CorvidForm".

Controls and Replaceable Parameters

Within the "form" tag (between the <form ...> and closing </form>) there can be HTML controls that allow the end user to select or enter values. This can include all the standard form input controls such as edit boxes, radio buttons, check Boxes, drop down lists, etc.

These are all added with the HTML <input>, <textarea> (larger blocks of text) and <select> (drop down lists) tags.

In the <input> tag, the type of control is determined by the "type=" value. For example:

<input type="text"...> will be an edit box

<input type="radio"...> will be an radio button

<input type="checkbox"...> will be a checkbox

In all cases, the "name" for the control MUST be the name of the Corvid Variable in square brackets. This is how the Corvid Servlet Runtime knows what variable should be assigned the input value.

For example, a text box control:

```
Temperature: <input type="text" name="[TEMP]">
```

would create an edit box labeled "Temperature:" and the value the user entered would be used to set the value of the Corvid variable [TEMP]

For radio buttons, the "value=" is used to specify the value for that radio button:

```
The color is
<input type=”radio”  name=”[COLOR]  value=”1”> Red
<input type=”radio”  name=”[COLOR]  value=”2”> Green
```

would create 2 radio buttons labeled “Red” and “Green”. The selecting the first would select the first value for the [COLOR] variable and the second would select the second value.

While controls can be added to templates in this way, it is very cumbersome and requires that the templates be carefully matched and coordinated with the system variables. Whenever the variables are edited, it would require changes in the templates and there would have to be a template for each variable. Corvid greatly simplifies this, and allows generic templates to be built, through replaceable parameters that can be used in the controls:

There are 5 replaceable parameters that greatly simplify building controls:

VARIABLE_PROMPT - Replace by the variable’s prompt text.

VARIABLE_NAME - Replaced by the variable’s name (Be sure to use in square brackets).

VARIABLE_VALUE_TEXT - Replaced by the variable’s value text (This is done in a CORVID_REPEAT loop that steps through each value for a List variable).

VARIABLE_VALUE_NUM - Replaced by the variable’s value number (also done in the CORVID_REPEAT loop).

With these replaceable parameters, the control for [TEMP]:

```
Temperature: <input type=”text” name=”[TEMP]”>
```

can be rewritten:

VARIABLE_PROMPT <input type=”text” name=”[**VARIABLE_NAME**]”>

When this control in the template is used to build a HTML form for [TEMP], the prompt and name for [TEMP] will be used, but it can just as easily be used for any other numeric, string or date variable. Corvid will automatically replace the parameters with the values appropriate for the variable being asked. Also, if the prompt or name is changed in the system rules, the change will automatically carry over to the HTML form built from the template.

The parameter “VARIABLE_PROMPT” in the template is just text in the HTML page and can be styled or formatted with CSS or any other design technique and the styling/formatting will apply to the text that is used to replace “VARIABLE_PROMPT”.

CORVID_REPEAT

List variables need to have multiple values either as a group of radio buttons /check boxes or in a list. To do this in a generic way that can be applied to variables with different numbers of values, there is the special Corvid command “CORVID_REPEAT”. This is added in the template as a HTML comment, “<!--CORVID_REPEAT-->”.

For a List variable, everything between “<!--CORVID_REPEAT-->” and the closing “<!--REPEAT_END-->” will be repeated for each value in the variable’s value list. The block of code will be used for the first value, then repeated for the second value, etc. T**he replaceable parameters VARIABLE_VALUE_TEXT and VARIABLE_VALUE_NUM will be replaced by the text and number of the value currently being added**.

With CORVID_REPEAT the the controls for [COLOR]:

```
The color is
<input type=”radio”  name=”[COLOR]  value=”1”> Red
```

```
<input type="radio"  name="[COLOR]  value="2"> Green
```

can be rewritten:

```
VARIABLE_PROMPT  <br>
<!--CORVID_REPEAT-->
<input type="radio" name="[VARIABLE_NAME]"
value="VARIABLE_VALUE_NUM"> VARIABLE_VALUE_TEXT
<!--REPEAT_END -->
```

When this template is applied to [COLOR] it will automatically build the same set of controls in the HTML form, but will build a set of radio buttons for as many values as the variable has and will automatically match the prompt and value text in the system. It can also be applied to any other List variable.

To use CORVID_REPEAT to build a list box control in a template, use:

```
VARIABLE_PROMPT <br>
<select name="[VARIABLE_NAME]">
<!-- CORVID_REPEAT -->
<option value="VARIABLE_VALUE_NUM"> VARIABLE_VALUE_TEXT
</option>
<!-- REPEAT_END -->
</select>
```

This will build a list of values for the end user to select from.

By using the replaceable parameters, it is easy to build a template that can be applied to many different variables and which will automatically pick up the text from the system. This greatly simplifies the maintenance on the user interface. Along with the template HTML code for the <form> and controls, the rest of the template can be anything that can be written in HTML. It can implement any design or look-and-feel. This allows creating a template that matches a site, but which can still be applied to many variables in a system.

Templates to Handle Different Types of Variables - CORVID_ASK

The last section described how to create “generic” templates that can be applied to all List variables, or all Numeric, String and Date variables, but Corvid also allows creating a template that has sections that will be included based on the individual variable type or name. This allows creating a single template that can be applied to all variables, with different controls for different types or even different controls for specific variables. The single template can implement a complex look-and-feel for all variable questions, while still being easy to maintain and automatically picking up the text from the system.

This is done with the CORVID_ASK command. Like the CORVID_REPEAT, this is added to the template HTML code as an HTML comment.

Everything in the template between a **<!-- CORVID_ASK VarID -->** and the closing **<!-- ASK_END -->** will ONLY be added to the HTML page that is built if the variable matches the variable ID, VarID.

The Variable ID can be broad (a type of variable) or a specific variable.

The allowed values for VarID are:

VARIABLE	All variables
LIST	All List variables
CONTINUOUS	All numeric, string and date variables
NUMERIC	All numeric variables
STRING	All string variables
DATE	All date variables
[VARNAME]	The specific variable varname
[VARMASK]	All variables fitting the mask pattern

If a mask pattern is used, the standard Corvid mask characters are used:

CHARACTER	MATCHES
?	Matches any character
*****	Matches the rest of the string
character	Matches itself
#	Matches any digit 0-9
{abc}	Matches any single character in the brackets { }
{X-Z}	Matches any single character between X and Z

A template file typically will have multiple CORVID_ASK sections, and MUST have a section that applies to each variable that will be asked using the template. For each variable that is asked using the template, **the first CORVID_ASK section which has a matching VarID will be used and all other CORVID_ASK sections will be ignored for that variable - even if they would also have matched**.

For example, if there were 3 sections in this order:

```
<!-- CORVID_ASK  [COLOR] -->
      Section 1 controls
<!-- ASK_END -->
 <!-- CORVID_ASK  [C*] -->
      Section 2 controls
<!-- ASK_END -->
<!-- CORVID_ASK  LIST-->
      Section 3 controls
<!-- ASK_END -->
```

The variable [COLOR] would use section 1 code, and ignore sections 2 and 3 since section 1 was already a match. A variable starting with "C", but not [COLOR], would use section 2 and ignore section 1 and 3. All list variables that did not start with "C" would use section 3 and ignore section 1 and 2.

One convenient way to make sure a template will work for all variables is to end with a <!-- CORVID_ASK VARIABLE --> section. This will apply to all variables that have not already matched a CORVID_ASK command in the page.

Care must be taken since List and Numeric variables typically need very different formats and controls for the questions. If a system has numeric, string, date, and List questions, it can be handled with 2 sections:

```
<!-- CORVID_ASK  CONTINUOUS -->
Section 1 controls
<!-- ASK_END -->
<!-- CORVID_ASK  VARIABLE -->
Section 2 controls
<!-- ASK_END -->
```

Section 1 will be used for all numeric, string and date questions. Section 2 will be used for all variables that are not Continuous, so it will be used for all List variables. Naturally, you might want to have more variation and have a separate section for each type.

To expand the template above to handle all variables that will be asked in a system:

```
<!-- CORVID_ASK  CONTINUOUS -->
VARIABLE_PROMPT  <input type=”text” name=”[VARIABLE_NAME]”>
<!-- ASK_END -->
<!-- CORVID_ASK  VARIABLE -->
VARIABLE_PROMPT  <br>
<!--CORVID_REPEAT-->
<input type=”radio” name=”[VARIABLE_NAME]” value=”VARIABLE_VALUE_NUM“>
VARIABLE_VALUE_TEXT
<!--REPEAT_END -->
<!-- ASK_END -->
```

Submit, Undo and Restart Buttons

Somewhere in the <form> tag, there must be a submit button. This is an <input> with the type set to “submit”:

```
<input type="submit" name="submitButtonName" value="   OK   ">
```

When this button is clicked, the value(s) selected are sent back to the Corvid Servlet Runtime. The “value” is the label that will appear on the button. This is set to “OK” in the default templates but can be changed to anything else, especially if the system is running in a language other than English.

The “name” is set to “submitButtonName”. It should NOT be changed since it is used in the default JavaScript for the page.

The submit button MUST appear in the page between the <form> and closing </form> tags.

The last screen in a system (typically a result screen) may have no "OK" button, and instead have only a RESTART button (described below) or a link back into your overall web site. This is recommended whenever there is no further processing to be done by the system, and the only action the user can take is to restart another session, or go to another page. This is typically only on a results template.

To do this, on the final screen a system will display, do NOT add an OK button, but only add the “Restart” button described below.

If the system needs to work in multiple languages, a double square bracket, [[ ]], replacement can be used for the name of the button. This variable would be set to a value appropriate to the language. Such as:

<input type="submit" name="submitButtonName"value="**[[OK_Btn_label.VALUE]]**">

In addition to the submit button that sends data, there can be 2 other special submit buttons: UNDO and RESTART. These also must be in form between the <form> and </form> tags.

UNDO Button

The UNDO button is **optional**. It will tell the Corvid Servlet Runtime to step back one question. This is the same as the BACK button in the Corvid Applet Runtime.

The HTML code for the UNDO button is:

```
<input type="submit" name="~UNDO" value=" Back ">
```

Note that this button also has the type set to “submit” and clicking it will send data back to the Corvid Servlet Runtime, however, in this case the values selected will be ignored and only the ~UNDO will be used.

The “value” is set to “Back”. This is the button label that will be displayed and can be changed to anything that is preferred.

The “name” is set to ~UNDO. This must NOT be changed. The ~UNDO is the special flag to tell the Corvid Servlet Runtime to process the input as an UNDO action.

The UNDO button can be placed anywhere on the screen. If UNDO is added to a template that will be used for many questions, Corvid will automatically disable the UNDO button for the first screen since there is nowhere to step back to. The button will be enabled in all screens where an UNDO is possible. Do not worry if the first question screen in a system has the UNDO disabled.

The UNDO action can also be achieved by the end user clicking the “Back” button on their browser. Corvid will automatically interpret this as an UNDO.

Corvid embeds a hidden value in each page to identify how to handle UNDO (or browser back) actions and the Corvid Runtime Servlet stores certain data on the server about the sessions. The amount of time that the server keeps this data available varies with the installation of the servlet engine and options that can be set. If a user waits a long time (typically over 30 minutes) and then tries to go back to an earlier Corvid session, the needed data may no longer be available and UNDO will not work. Likewise, bookmarks for a specific question will not work after the session data is gone and will instead require restarting the system.

RESTART Button

The RESTART button is **optional**. It will tell the Corvid Servlet Runtime to restart the system from the beginning of the starting Command Block. This is the same as the RESTART button in the Corvid Applet Runtime.

The HTML code for the UNDO button is:

```
<input type="submit" name="~RESTART" value=" Restart ">
```

Note that this button also has the type set to “submit” and clicking it will send data back to the Corvid Servlet Runtime, however, in this case the values selected will be ignored and only the ~RESTART will be used.

The “value” is set to “Restart”. This is the button label that will be displayed and can be changed to anything that is preferred.

The “name” is set to ~ RESTART. This must NOT be changed. The ~RESTART is the special flag to tell the Corvid Servlet Runtime to process the input as a RESTART action.

The RESTART button can be placed anywhere on the screen. If RESTART is added to a template that will be used for many questions, Corvid will automatically disable the RESTART button for the first screen since there is nowhere to step back to. The button will be enabled in all other screens. Do not worry if the first question screen in a system has the RESTART disabled.

Multiple Submits and JavaScript

When data is sent back to the Corvid Servlet Runtime, it can take a few moments for it to be processed and a new HTML page sent back to the end user. If the user clicks the “Submit” button again while Corvid is processing the data, it can produce an error since Corvid has already started using the data from the first “submit”. This will result in Corvid displaying a “Multiple Submits” error and the end user will have to step back and try again. This can be annoying for the end user and is easy to prevent with a little JavaScript which is already included in the default templates.

CSS in the Default Template

The Corvid templates use CSS to set the style and position of text and controls. The default question template uses:

```
<style type="text/css">
.VariablePrompt {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 14px;
        color: #003366;
        font-weight: bold;
        text-indent: 50px;
}
.VariableValue {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 12px;
        color: #003366;
        font-weight: bold;
        text-indent: 75px;
}
.VariableEditBox {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 12px;
        font-weight: bold;
        width: 200px;
        text-indent: 75px;
}
</style>
```

The styles control the font and positions of the prompt (VariablePrompt), font and potions of the List variable values (VariableValue) and size and position of the edit box for other types of variables (VariableEditBox). An easy way to modify the look of the page is to change the CSS.

Images in the Template

A question template file often will incorporate image files for backgrounds or other design elements. Normally an HTML page is a physical page on a server, and images can be put in the same folder or a subfolder. This allows the images to be referenced using the location of the page as a base address. For example, if a page uses an image named "my_image.jpg" and that image is in the same folder (directory) as the page, the page HTML can just use "my_image.jpg" and the Browser will automatically look in the same folder as the page. However, a page displayed from the Corvid Runtime Servlet is created dynamically and does not really have a physical location on the server. Consequently, image files cannot be specified by relative location to a page, and requires a more complete URL address. In addition, the template file usually will not be in a location on the server that allows Web browsing (though it can be), however, the image files MUST be in a section that is accessible via Web browsers.

The image files in a template can be referred to in several ways:

Full URL

If you know where the image is located on a server, just include the full URL to that image. This requires that the image be in a location that can be referred to by a URL - entering the URL in a Web browser must display the image.

For example:

```
<IMG SRC="http://www.mysite.com/images/myImage.jpg">
```

Use BASE to set a Base URL for Images

If all the images are in one folder (or subfolders of a folder) you can use the

```
<BASE href="…">
```

tag in the template. The <BASE> tag indicates the location to use for all images and links. Any image file or link that is not specified by a full URL starting with "http://" will use this base address as the starting location. The BASE tag must occur in the HEAD section of the template - that is between the <HEAD> and </HEAD> tags. The base URL can be hard coded such as:

```
<BASE href="http://www.mysite.com/images/">
```

Then the image can be referenced from that base:

```
<IMG SRC="myImage.jpg">
```

Corvid_LINK_BASE

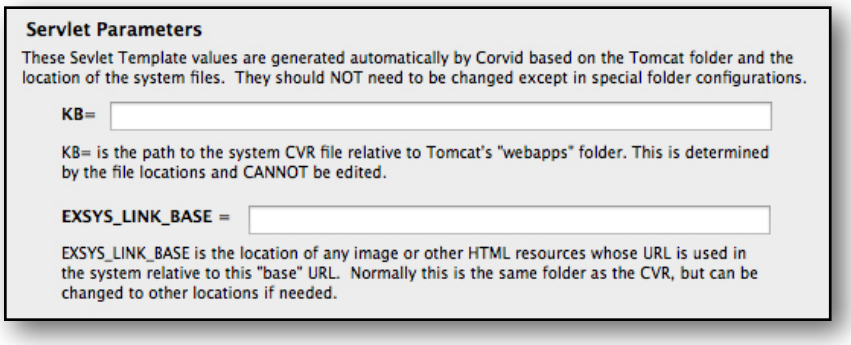
Corvid provides a way to code the address of the linked images using the replaceable parameter “CORVID_LINK_BASE”. This is another of the replaceable parameters that can be used in the template and Corvid will replace it with a value set in the system.

If a template has:

```
<BASE href="CORVID_LINK_BASE">
```

The <BASE> value will be set to the value set in the system for “CORVID_LINK_BASE”.

The CORVID_LINK_BASE parameter is set from the Tomcat Setup window.



If a system has multiple templates, using this approach makes it easy to change all the pages that will be built from the templates simply by changing one value in the system. It also makes it easier to use the same templates for multiple systems since the <base> value does not have to be hard coded.

Other HTML Content in the Template

A template must have the required Corvid sections described above to function as a template, but any other content in the template is just passed through to the HTML page it creates. This means that ANYTHING that could be in a legal HTML page can be added to the template and it will appear in the resulting HTML page.

Corvid only parses its special commands. Anything that does not have syntactical meaning to Corvid is passed through unchanged. The template can include any HTML commands, HTML5, CSS3, JavaScript, Ajax, plugins, etc. - anything the browser supports.

Single Variable Templates

Corvid provides various ways to implement more advanced features using templates. These provide a large degree of flexibility and if something very special is needed, a “template” can always be created for the individual question.

A “template” for a single variable is not really a template since it will not be used for multiple variables. It can be designed to ask the specific variable any way that is required. The one “template” parameter that is required, even when only for a single variable is the action=”CORVID_SERVLET” in the <FORM> tag:

```
<form method="post"  action="CORVID_SERVLET" >
```

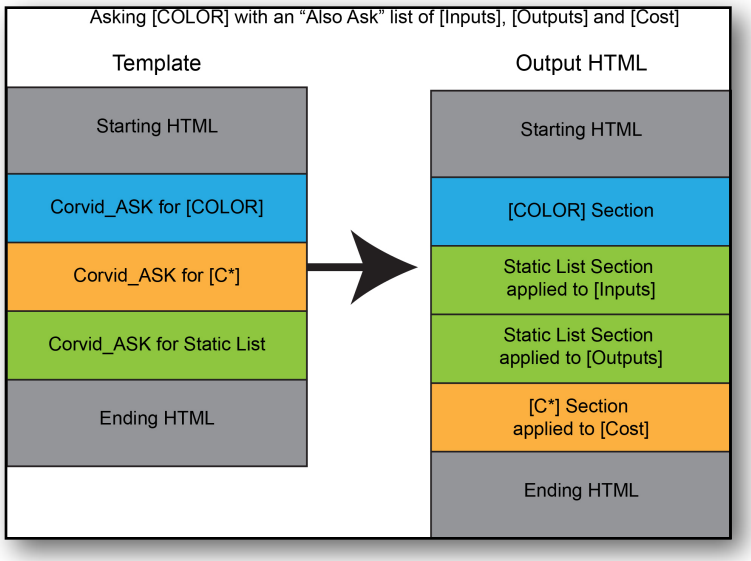
Other parts of the form can be modified or hard coded, but the action MUST remain CORVID_SERLVET which will be replaced by the actual value by the Corvid Servlet Runtime. This cannot be hard coded into a page because the actual value is not known until runtime.

It is also recommended that the “method” remain “post” (It will work with “get”, but there is no real benefit to the change and it limits the amount of data that can be sent. Since Corvid sends some hidden data to keep track of the session, it is best to not limit the amount of data that can be sent, even if the controls on the page do not send much data on their own.

16.3 “Also Ask” and Templates

The Servlet Runtime supports the "Also Ask" feature of Corvid, which allows multiple questions to be easily asked on one screen. The “Also Ask” list is set in the same way as with the Applet Runtime. When the selected variable is asked, each variable in the Also Ask list will be asked on the same screen.

In the servlet version, all questions will be asked using the same template associated with the initial variable being asked. That template MUST have sections appropriate for each variable that will be asked in the "Also Ask" group. That means there must be a CORVID_ASK section that will match each variable in the Also Ask list.



The CORVID_ASK sections for each Also Ask variable in the template screen can be in any order. The initial variable will be asked with its associated CORVID_ASK section, followed in order by each of the Also Ask variables asked with their associated CORVID_ASK sections.

For example, using the CORVID_ASK section:

```
<!-- CORVID_ASK  [COLOR] -->  Section 1  <!-- ASK_END -->
<!-- CORVID_ASK  [C*] -->  Section 2    <!-- ASK_END -->
<!-- CORVID_ASK LIST-->  Section 3    <!-- ASK_END -->
```

If a system asks the variable [COLOR] which has an "Also Ask" list of the variables [INPUTS], [OUTPUTS] and [COST]. Where [INPUTS] and [OUTPUTS] are list variables, the screen would have:

[COLOR] asked Section 1 used to ask [COLOR]

[INPUTS] asked with Section 3

[OUTPUTS] asked with Section 3

[COST] asked with Section 2

The template for the initial variable to ask is used for all variables in the Also Ask list - even if that is not the template associated with those individual Also Ask variables. This allows a variable to be asked in different ways. If the variable [COST] is asked as an Also Ask from [COLOR], it will use the section of the template associated with [COLOR] that matches [COST]. If no input is provided for [COST], and the value is needed by the system, it would be re-asked using the template associated with [COST]. The second time [COST] is asked, you could use a different template that reminds the user that this input was not previously provided. (This ability to ask a question different ways can easily be done in the servlet version, but is not easy to do using applets.)

16.4 Control Options to Ask List Variables

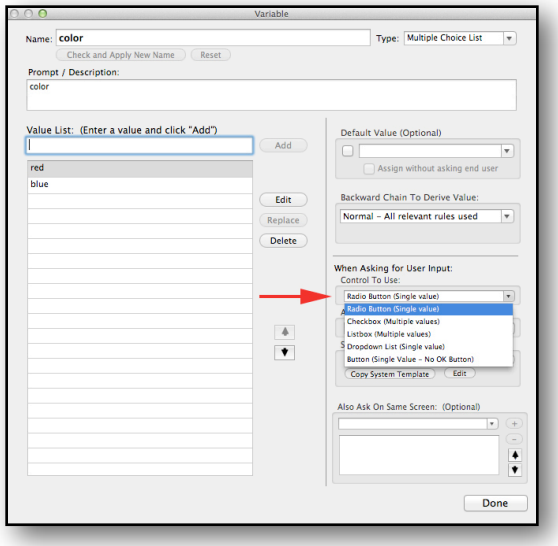
List variables have more options in the controls that can be used than any other type of variable. They can be asked with radio buttons and check boxes, but when there are many possible values, or the screen space is limited, list controls are a good alternative. List controls can also be asked using buttons. As soon as the user clicks on a button the value is sent to Corvid, without having to click the submit button.

Checkbox vs Radio Button - Automatic Switching

In many systems, some variables are only allowed to have a single value and should use radio buttons (or lists), and should not use checkboxes that may allow more than one value to be selected at the same time.

Other variables can accept multiple values and should use checkboxes. To allow the template to be generic, **always design templates with checkboxes unless radio buttons should always be used**. For those variables that are limited to only a single value, Corvid will automatically convert the checkbox controls to radio buttons for that variable. Variables that can have multiple values will use the checkboxes.

A variable is set to allow only a single value from the Variable window by clicking the Options tab. Set the "Maximum Number of Value that be Assigned" to "Single Value". Then any CORVID_ASK section that is specified for "checkbox" will automatically have it converted to "radio".



If the control is set for:

```
type="checkbox"
```

and “Radio Button (Single value)” is selected, Corvid will automatically convert the “checkbox” to “radio”.

NOTE: The “Ask With” tab options for specifying other controls to use apply ONLY to the Corvid Applet Runtime and do not change the control used by the Servlet. The servlet requires the control to be set in the template, and only uses the single/multiple option to convert between radio buttons and check boxes

16.5 List Control Layout

To ask for the value of all List variables with one value per line, add a
 in the CORVID_REPEAT:

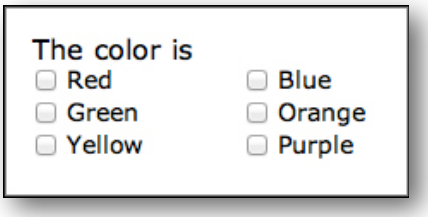

```
<!-- CORVID_ASK LIST -->
VARIABLE_PROMPT <BR>
<!-- CORVID_REPEAT -->
<input type="checkbox" value="VARIABLE_VALUE_SHORT"
name="[VARIABLE_NAME]">VARIABLE_VALUE_TEXT <BR>
 <!-- REPEAT_END -->
<!-- ASK_END -->
```

if instead you wanted 2 columns of values, the checkboxes could be put in a table. This could be hard coded for a variable or something like this could be used:

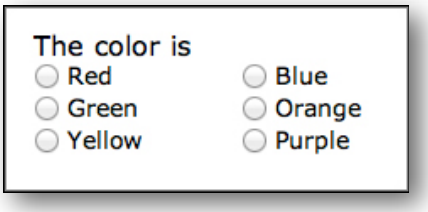
```
<!--CORVID_ASK LIST -->
VARIABLE_PROMPT
<div align="center">
<table border="0" cellpadding="0" cellspacing="0" width="90%">
<tbody>
<tr>

<!--CORVID_REPEAT-->
    <td width="50%">
    <input value="VARIABLE_VALUE_NUM" name="[VARIABLE_NAME]"
    type="checkbox">  VARIABLE_VALUE_TEXT
    </td>
```

```
    <!--CORVID_IF ((VARIABLE_VALUE_NUM % 2) == 0) -->
          </tr><tr>
    <!-- END_IF -->

<!--REPEAT_END-->

</tr>
</tbody>
</table>
</div>
<!--ASK_END-->
```

Here the trick is that to have a table, there needs to be a "</tr><tr>" to end one row and start another but only after every second value. This is done with the CORVID_IF, which checks to see if the value of VARIABLE_VALUE_NUM is divisible by 2. IF it is, the </tr><tr> is added. Remember, the CORVID_REPEAT will loop once for each of the variable's values setting the VARIABLE_VALUE_NUM and VARIABLE_VALUE_TEXT for that value.



If [COLOR] was set to only allow a single value, the "checkbox" would automatically be converted to "radio" and it would look like:

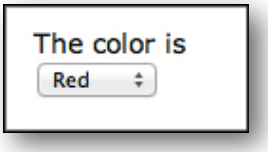


Drop Down Lists

Drop down lists are a very convenient way to set the value for a List when there are a large number of values or screen space is limited. Lists are built with the HTML <select> and <option> tags:

```
<select name="[VARIABLE_NAME]" size="#">
<!-- CORVID_REPEAT -->
<option value="VARIABLE_VALUE_NUM"> VARIABLE_VALUE_TEXT
</option>
<!-- REPEAT_END -->
</select>
```

This will create a list of values. The "size" value is the number of rows that will be displayed in the control. If the size is set to 1 (size="1"), then the control will be a drop down list with only the selected value displayed. If the size is set to a higher value, the control will be a list with the number of rows displayed equal to the size value and a scroll bar if needed.



If the same code had the size= parameter changed to size="4", a list control would be produced:

To allow the end user to select multiple values from the list, "multiple" should be added after the size= to indicate that multiple values can be selected from the list.

```
<select name="[VARIABLE_NAME]" size="5" multiple>
```

If the variable only allows a single value to be selected, Corvid will automatically remove the "multiple" and build a list control that only allows a single value. Adding "multiple" makes a more generic control that will work correctly for all List variables. If all the variables in a system only allow a single value the "multiple" option can be left off.

Buttons to Ask Questions

Buttons provide one more way to build questions screens for List variables.

Most of the question controls require that the user select a value from a list or by checking an item, and then click the “OK” button. That approach has the advantage that the end user can review their input and make changes before submitting it. However, for some systems, a better end user interface is one that just allows the user to click on a button and have the system immediately take the input. This is especially true if the system asks multiple Yes/No or True/False questions, or systems running on iPads/iPhones.

With buttons, when the end user clicks on a button, the value is immediately sent to Corvid, and no “submit” button has to be clicked. This should be used when there is only a single question asked on the screen.

Buttons can be added using the <input> tag with the type=”submit” . However, unlike the standard “submit” button, this one has the “value” set to the expression:

```
value="[VARIABLE_NAME]=VARIABLE_VALUE_NUM”
```

When the parameters are replaced by actual variable values, his would be converted to something more like:

```
value="[Color]=2”
```

Note, this is different than other controls where the “value” is just the value number of short text. Here it is an expression to assign the value to the variable.

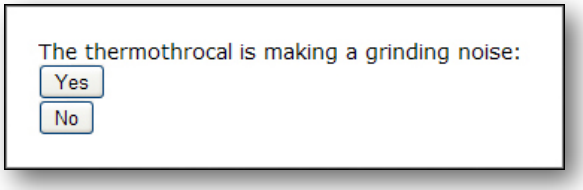
The “name” for the <input> set to the text for the value. This will become the label on the button.

```
name="VARIABLE_VALUE_TEXT"
```

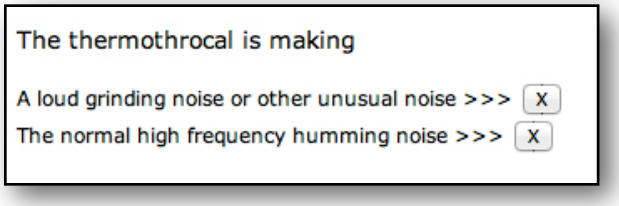
The HTML to ask any List with buttons would be:

```
VARIABLE_PROMPT<BR>
<!-- CORVID_REPEAT -->
<input type="submit"
value="[VARIABLE_NAME]=VARIABLE_VALUE_NUM" name="VARIABLE_VALUE_TEXT">
 <!-- REPEAT_END -->
```

For example:



If the text is too long to put on the button, it can be put next to the button, with the button having an "*", image or some other generic label. This can be done with:



```
VARIABLE_PROMPT<BR>
<!-- CORVID_REPEAT -->
VARIABLE_VALUE_TEXT
<input type="submit"
value="[VARIABLE_NAME]=VARIABLE_VALUE_NUM" name="X">
<!-- REPEAT_END -->
```

SECURITY NOTE: Variable values are sent back to the servlet using POST. Corvid does NOT encrypt the information itself and if the user is providing highly sensitive password information, a secure server connection (https) should be used.

16.6 Control Options to Ask Numeric, String and Date Variables

Simple One-line Text Edit Box

When asking for the value of Numeric, String or Date variables, the end user is typically presented with a prompt and an edit box where they can input the value. f their input is reasonable short, this can be done with:

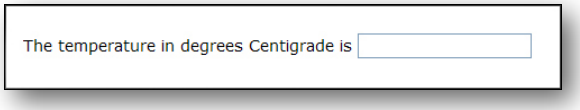
```
<input type="text" name="[VARIABLE_NAME]">
```

This allows a single line of text, which will be assigned to the variable. The tag allows an optional size="#" parameter that sets the size of the edit field, where # is a numeric value. The positioning of the edit field is done using HTML formatting commands or CSS outside of the tag. The prompt for the variable can be displayed to the left of this tag or above it and should use the VARIABLE_PROMPT replaceable parameter.

For example, a template section that would work for all numeric variables:

```
<!-- CORVID_ASK NUMERIC -->
VARIABLE_PROMPT <input type="text" name="[VARIABLE_NAME] size="12">
<!-- ASK_END -->
```

Using this to ask for the temperature would look like:



Larger, Multiline Edit Box

For a larger, multi-line edit box, use the <textarea> control.

```
<textarea name="[VARIABLE_NAME]" cols="#" rows="#">  </textarea>
```

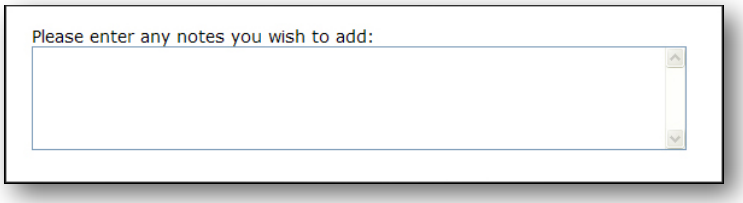
This allows inputting multiple lines of text, and handles scrolling etc. The value entered will be assigned to the variable. The cols="#" parameter that sets the number of columns in the edit box (width) and rows="#" sets the number of rows, where # is a numeric value. The positioning of the edit field is done using HTML formatting commands or CSS outside of the tag. The prompt for the variable can be displayed to the left of this tag or above it and should use the VARIABLE_PROMPT replaceable parameter.

Note that the tag requires the closing </textarea> tag to mark the end. If you wish to have text already in the edit box when it is displayed, put it before the </textarea> marker. In most cases you will want the edit field will be blank and there will be no text between <textarea…> and </textarea>.

To ask for the value of all string variables:

```
<!-- CORVID_ASK STRING -->
VARIABLE_PROMPT <BR>
<textarea name="[VARIABLE_NAME]" cols="60" rows="5">  </textarea>
<!-- ASK_END -->
```

Using this to ask for a note would look like:



The <textarea> control can also be used to display content generated by the system and in a variable.

For example you could display a comment generated by the system to see if the user wishes to add to it. The system generated comment is in the variable [SysComment] which is included with double square bracket, [[ ]], embedding. This is put before the closing </textarea>.

The user's modified note will be sent to the variable [UserNote] (In this case hard coded into the template):

```
Please enter any notes you wish to add:
<textarea name="[UserNote]" cols="60" rows="5">  [[SysComment]]
</textarea>
```

Password Edit Box

If a system needs an edit box for a password or other text that should not be displayed on the screen, use:

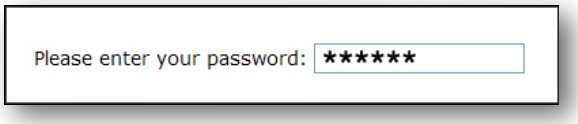
```
<input type="password" name="[VARIABLE_NAME]">
```

This allows a single line of text, which will be assigned to the variable. It is similar to the simple edit box above, but the edit box will echo back "*" or some meaningless symbol instead of the user's input. This hides the text that is typed in and can be used for passwords. The tag allows an optional size="#" parameter that sets the size of the edit field, where # is a numeric value. The positioning of the edit field is done using HTML formatting commands or CSS outside of the tag. The prompt for the variable can be displayed to the left of this tag or above it and should use the VARIABLE_PROMPT replaceable parameter.

For example, to ask for the value of all variables starting with "PASSWORD":

```
<!-- CORVID_ASK [PASSWORD*] -->
VARIABLE_PROMPT <input type="password"  name="[VARIABLE_NAME]>
<!-- ASK_END -->
```

Using this to ask for a password would look like:



16.7 Using Image Maps to Ask Questions

In most cases, the HTML screens that ask a user to answer a question are forms that use POST to return the data to the servlet. However, a system can also use simple HTML links to send data to the servlet. This includes image maps and individual images or text that have associated HTML links.

The key to doing this is creating a link that calls back to the Corvid Servlet Runtime, with information that identifies the session and the data to be returned. Corvid provides a simple replaceable parameter that allows you to do this easily:

```
CORVID_SERVLET_GET
```

The parameter CORVID_SERVLET_GET will be replaced by a call to the Corvid Servlet Runtime along with information that will identify the place in the session. The link back to the Corvid Servlet Runtime must NOT be hard coded, since the additional information will not be known until the system is run.

The GET approach will be used to send the data. Unlike the POST approach normally used in the templates, this approach adds the data to the URL that is created. Corvid will automatically add some additional information, which is normally passed as hidden fields in the POST data sent by the forms.

To use CORVID_SERVLET_GET:

- Add a normal URL link to an image map, image or text that should returns a value to Corvid.
- Make the link "CORVID_SERVLET_GET " followed by the name of the variable in [ ], an "=", and the value to assign. If there are multiple values, separate them with "&". There should not be any spaces between data. If there are spaces or other characters in the value that require URL encoding, they should be encoded.

For example, a link off an image (or section of an image map) that sets the value of [X] to 5 might be:

```
<a href="CORVID_SERVLET_GET [X]=5">
```

Corvid will replace the CORVID_SERVLET_GET parameter with the correct address, plus a "?", the session identification, an "&", and the data you provide.

For List variables, the value assigned can be a numeric value that is the number of the value, or the text of the value. For Numeric variables, the value would be numeric. For string variables, the value would be a string. It would not need to be in quotes, but might require URL encoding if it includes spaces or other characters that need to be encoded.

For a generic template that asks a question and assigns the value 5, you could alternatively use:

```
<a href="CORVID_SERVLET_GET [VARIABLE_NAME]=5">
```

This would be unusual since almost all cases that would use this technique will require a specific image or image map. Normally image map screens are not used generically and are associated with only a single variable, so using the actual variable name is more standard. However, using VARIABLE_NAME in templates does allow the variable's name to be changed in the system without having to edit the template.

If you wished to return data for more than one variable, you can make a list of values separated by "&", for example to set [X] to 5 and [Y] to 2:

```
<a href="CORVID_SERVLET_GET [X]=5&[Y]=2">
```

Remember not to add any spaces and to URL encode any characters that require it.

An example of a page that uses an image map to ask a question is:

```
<html>
<head>
<title>Image Map Demo</title>
</head>
<body>
<BASE href="Corvid_LINK_BASE">
<div align="center">
<img src="MyImage.gif" width="620" height="120" border="0"
```

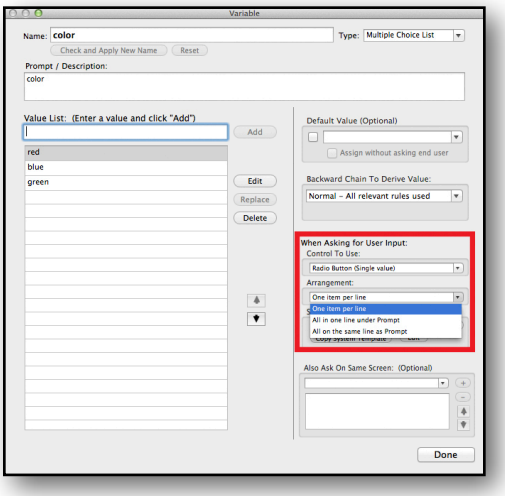
```
usemap="#map1162e6c">
<map name="map1162e6c">
<area shape="rect" coords="497,29,613,115"
href="CORVID_SERVLET_GET [X]=1">
<area shape="rect" coords="377,31,488,114"
href="CORVID_SERVLET_GET [X]=2">
<area shape="rect" coords="255,29,368,114"
href="CORVID_SERVLET_GET [X]=3">
<area shape="rect" coords="138,27,249,111"
href="CORVID_SERVLET_GET [X]=4">
<area shape="rect" coords="12,27,126,114"
href="CORVID_SERVLET_GET  [X]=5">
</map>
</div>
</body>
</html>
```

This image map has 5 specified regions with associated links. A click on any of the regions will trigger the link and set the value of the variable [X].

Using image maps is quite easy with an HTML editor that allows defining the regions and links. This allows very advanced user interfaces to be created. An alternative to an image map is to have several image files (jpg or gif) on a page with the same type of CORVID_SERVLET_GET link off them. A click on any of the images would set the associated value. This provides another way to ask questions with images. Any URL link on a page can use the CORVID_SERVLET_GET approach to send data to Corvid.

16.8 Special Options for Generic Question Templates

The default question template in Corvid is designed to work with the options set for how a variable should be asked. These are set in the “When Asking for User Input” controls for the type of control to use (radio button, list, etc) and the arrangement relative to the prompt.



The Default Question template is designed to convert the options set into the HTML code. This is done using the CORVID_IF command with special parameters:

#ARRANGE=ONEPERLINE
#ARRANGE=UNDERPROMPT
#ARRANGE=SAMEASPROMPT

#CONTROL=RBCB
#CONTROL=DROPDOWN
#CONTROL=LIST
#CONTROL=BUTTON

Parameter	Boolean Value
#ARRANGE=ONEPERLINE	TRUE if Arrangement is set to one item per line, otherwise FALSE.
#ARRANGE=UNDERPROMPT	TRUE if Arrangement is set to under the prompt, otherwise FALSE.
#ARRANGE=SAMEASPROMPT	TRUE if Arrangement is set the same line as the prompt, otherwise FALSE.
#CONTROL=RBCB	TRUE if the Control is set to radio button or check box, otherwise FALSE. (Corvid converts between radio buttons and check boxes automatically)
#CONTROL=DROPDOWN	TRUE if the Control is set to a drop down list, otherwise FALSE.
#CONTROL=LIST	TRUE if the Control is set to a List, otherwise FALSE.
#CONTROL=BUTTON	TRUE if the Control is set to a Button, otherwise FALSE.

These parameters must be used exactly as described above - capitalized with no spaces.

Looking at the Default Question templates shows how these are used to conditionally add sections of HTML code to handle the various options.

In general, when custom templates are created for a variable, they will not need to be “generic” and will just implement a design for that specific variable and the options will be “hardcoded” rather than picked up from the CVR file. In that case, these options are not needed. However, designing a new “custom” template that is generic and can pick up the parameters set in the system should use these parameters to handle the various possible cases.

Appendix A - Operators and Functions

A.1 Expression Operators

The standard equal, greater than, less than operators, etc are used:

= or ==	Equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal
~=	Approximately equal

The approximately equal boolean operator has 2 meanings depending on the type of expressions being compared:

String ~= String

Performs a case insensitive string comparison of the 2 string expressions. (The normal String = String comparison is case sensitive) (The UCase() conversion function can also be used to do case insensitive comparisons)

For example, "hello world" ~= "Hello WORLD" is true

"hello world" = Hello WORLD" is false

Numeric ~= Numeric

Tests if two numerics are approximately equal. It tests if the difference of the 2 numbers is within .005% of their sum. The actual algorithm allows negative numbers too. This allows a comparison of calculated numbers where roundoff error may have made them very slightly different.

For example, 1000000000 ~= 1000100001 is true

1000000000 ~= 1000100006 is false

A.2 Functions

Corvid supports a wide range of functions that can be used to build the expression needed in systems. These range from the standard trigonometric functions found in most computer languages to specialized functions for parsing strings and dates.

Each function returns a value of a particular type and must be used in an expression where that type is syntactically correct. For example, the sine function SIN(X) will return a numeric value and the argument "X", must also be a numeric value. SIN(.5) is legal, but SIN("ABC") is not since "ABC" is a string rather than a numeric value. Likewise, [X]=1+SIN(.5) is legal, but [S]="ABC" + SIN(.5) is not since the expression would require a string value and SIN is numeric.

The argument of a function can be a complex expression involving other functions and Corvid variables, provided the overall expression is the correct type. For example:

SIN(1 + (([X] + [Y]) / [Z]))

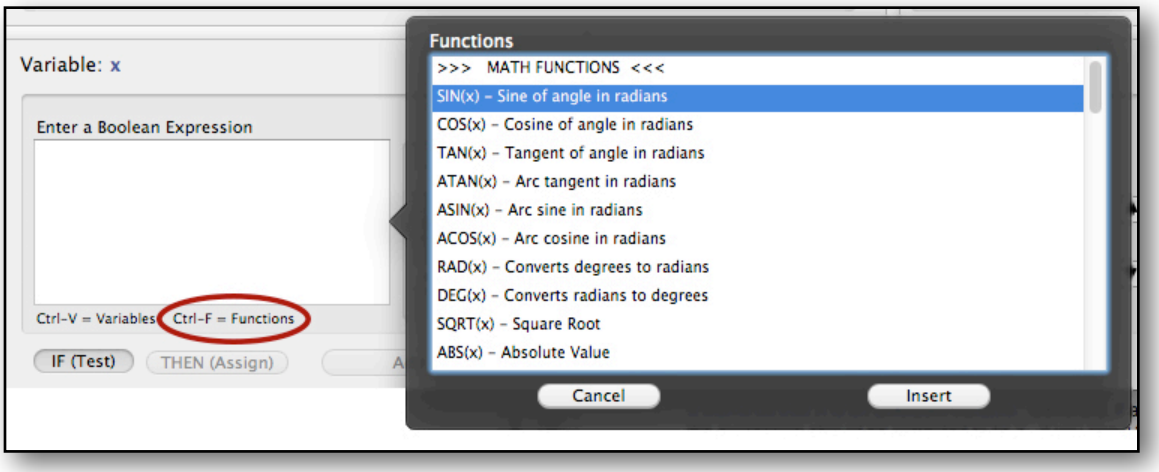
SIN(SQRT([X]))

The argument to a function must be in parenthesis. Some functions take more than one argument and the individual arguments must be separated by commas.

Most windows for entering expressions have a “Functions” button. Clicking the button displays a list of functions supported in Corvid. Simply select the function needed and click OK, or double click on the function. The function prototype will be added to the edit box at the current cursor and can then be edited.

Function Popup

Edit boxes where functions are likely to be used, display a popup list of the available functions. Hold the **Control key down and press the F key** - to display the list of functions along with a brief description.



To add a function, just double click on it in the popup. The function will be added to the expression with placeholders for the arguments that the function takes. Just edit the arguments to whatever is needed in your situation. Any edit box that supports the Function popup will be marked “Ctrl-F=Functions” under the edit box to remind you.

A.3 Numeric Functions

SIN(x) Sine of angle in radians

Returns: Numeric Argument: Numeric

Example: SIN(.5) SIN([X] + 1)

COS(x) Cosine of angle in radians

Returns: Numeric Argument: Numeric

Example: COS(.5) COS([X])

TAN(x) Tangent of angle in radians

Returns: Numeric Argument: Numeric

Example: TAN(.5) TAN([X])

ATAN(x) Arc tangent

Returns: Numeric value in radians Argument: Numeric

Example: ATAN(.5) ATAN([X])

ASIN(x) Arc sine

Returns: Numeric value in radians Argument: Numeric

Example: ASIN(.5) ASIN([X])

ACOS(x) Arc cosine

Returns: Numeric value in radians Argument: Numeric

Example: ACOS(.5) ACOS([X])

SQRT(x) Square Root

Returns: Numeric Argument: Numeric

Example: SQRT(4) SQRT([X])

ABS(x) Absolute Value

For positive number, return the value. For negative values, returns the value converted to a positive

Returns: Numeric Argument: Numeric

Example: ABS(4) = 4 ABS(-4) = 4

EXP(x) e raised to the power x

Returns: Numeric Argument: Numeric

Example: EXP(4) EXP([X])

LN(x) Natural Logarithm (log base e)

(For log base 10, use LOG(x))

Returns: Numeric Argument: Numeric

Example: LN(4) LN([X])

LOG(x) Base 10 Logarithm

(For log base e, use LN(x))

Returns: Numeric Argument: Numeric

Example: LOG(4) LOG([X])

FLOOR(x) Highest integer value less than x.

(See comparison with INT(x) below)

Returns: Numeric Argument: Numeric

Example: FLOOR(4.3) FLOOR([X])

INT(x) Integer part of x

Returns: Numeric Argument: Numeric

For positive numbers, the INT() and FLOOR() functions give the same value. For negative numbers, FLOOR() is the first integer value less than the argument, while INT() is the integer part of the negative number. FLOOR(-2.5) = -3 while INT(-2.5) = -2.

Example: INT(4.3) INT([X])

FRAC(x) Fractional part.

Applies to positive or negative numbers.

Returns: Numeric Argument: Numeric

Example: FRAC(4.5) = .5 FRAC(-2.1) = .1

ROUND(val, fraction) Round a value

Rounds a value off to the nearest integer, or if a fraction is specified, to the nearest value of that fraction.

Returns: Numeric Arguments: Numeric

val The number to round

fraction An optional parameter

The fractional value increment to round off to. The fraction does not have to be less than 1, but it does have to be positive and non-zero.

ROUND(x) Rounds x to nearest integer

ROUND(x, y) Rounds x to the nearest multiple of y

Examples:

ROUND(10.6)=11

ROUND(10.6, 0.5)=10.5

ROUND(10.7, 0.25)=10.75

ROUND(17.339, 0.1)=17.3

ROUND(256, 100)=300

RANDOM() Generates a random number between 0 and 1

Returns: Numeric Argument: None

Example: RANDOM()

MIN(x, y, z,...) Minimum of list of values.

Returns the lowest value from the list. There can be any number of numeric arguments separated by commas

Returns: Numeric Argument: List of numeric values separated by commas

Example: MIN(5, 3, 1, 7) = 1

MAX(x, y, z,...) Maximum of list of values.

Returns the largest value from the list. There can be any number of numeric arguments separated by commas

Returns: Numeric Argument: List of numeric values separated by commas

Example: MAX(5, 7, 1, 3) = 7

RAD(x) Convert Degrees to Radians

Returns: Numeric Argument: Numeric

Example: RAD(234)

NUM(s) Convert a string to a number.

The string must be a string representation of a number in any of the supported formats. This is can be used when a string (such as one obtained from a tokenize() call) needs to be used as a numeric in a calculation.

Returns: Numeric Argument: String

Examples: NUM(“123”) = 123

NUM(“1.5e3”) = 1500

NUM(“abc”) = illegal

A.4 String Boolean Tests

The boolean comparison operators have a different meaning when applied to strings. The comparison is case **insensitive** (upper and lower case letters are equivalent). For strings A and B, the operators are:

A = B	True if string A is the same as string B
A >= B	True if A is the same or alphabetically greater than B
A <= B	True if A is the same or alphabetically less than B
A > B	True if A is alphabetically greater than B
A < B	True if A is alphabetically less than B
A != B	True if string A is not the same as string B
A <INSTR> B	True if string A is a substring of string B

A.5 String Functions

Corvid string functions are primarily for the parsing of strings that contain information needed by a system, or to build reports.

LEFT(s, n)
Left n characters of string s
Returns: String Arguments: s - String, n-numeric
Example: LEFT(“abcde”, 3) = “abc”

RIGHT(s, n)
Right n characters of string s
Returns: String Arguments: s - String, n-numeric
Example: RIGHT(“abcde”, 3) = “cde”

MID(s, x, n)
Middle n characters of string s starting at character number x
Returns: String Arguments: s - String, x,n-numeric
Example: MID(“abcdefg”, 3, 2) = “cd”

LEN(s)
Length of string s
Returns: Numeric Argument: String
Example: LEN(“abcde”) = 5

UCASE(s)
Convert string s to all upper case
Returns: String Argument: String
Example: UCASE(“abc”) = “ABC”

LCASE(s)
Convert string s to all lower case
Returns: String Argument: String
Example: UCASE(“ABC”) = “abc”

A <INSTR> B

Returns the position of string A in string B, or 0 if string A does not occur in string B. This can be used as a boolean test since 0 is equivalent to “False” and any non-zero value is equivalent to “True”. It can also be used in numeric expressions that need the position of string A in string B. The first character of string B has a value of 1.

The comparison is case sensitive – a lower case letter will NOT match an upper case letter. To do a comparison that is not case sensitive, force both strings to upper case with the UCASE() function.

(e.g. UCASE([s]) <INSTR> UCASE([t]) would check to see if the string value of [s] is in [t] ignoring case.)

This can be used anywhere a Boolean expression is allowed.

Both Examples:

"hello" <instr> "hello world"	would return a value of 1 or “True”
“x” <instr> “abc”	would return a value of 0 or “False”
“a” <instr> “ABC”	would return a value of 0 or “False” – cases do not match
“abc” <instr> [s]	would return the position of “abc” in the value of variable [S] or 0 if “abc” does not occur in the value of [S]. This can be use to test if “abc” is in the value of [S].

FINDCHR(sourceStr, searchCharSet)

Returns the numeric index of the first character in the sourceStr string that is a character in the searchCharSet string.

Returns: String Arguments: String

sourceStr The string to search

searchCharSet A string containing the characters to search for

Returns the numeric index of the first character in **sourceStr** that is in **searchCharSet**. If no character from **searchCharSet** is found, 0 (False) is returned. The index of the first character in **sourceStr** is 1.

Both **sourceStr** and **searchCharSet** are strings and should either be the value of a string variable, a string expression or a string in " ". The character match is case sensitive.

Examples:

FINDCHR("abcdefghij", "bx") returns 2

FINDCHR("abcdefghij", "xyz") returns 0

FINDCHR("abcdefghij", "df") returns 4

REVERSE(sourceStr) Reverse the order of the characters in a String

Returns the sourceStr reversed. This makes it possible to perform actions on the right side of the source string using a function that acts on the left side of the source.

Returns: String Arguments: String

sourceStr The string to reverse. It should be either the value of a string variable, a string expression or just a string in " "

Example: REVERSE("abc") returns "cba"

MATCHES(sourceStr, patternStr) Test if a String Matches a Pattern

Returns TRUE if the **sourceStr** can be matched against the **patternStr**. If the pattern does not match, it returns FALSE. In a numeric context TRUE is 1 and FALSE is 0.

Returns: String Arguments: String

This uses the same pattern matching syntax as when creating a mask for a variable or block name.

Masks are specified by a string that indicates which character(s) is acceptable:

Character	Matches
?	Any character
*****	The rest of the string
character	Itself
#	Any digit 0-9
{abc}	Any character in the { }
{X-Z}	Any character between X and Z

Examples:

MATCHES("abc123", "a?c###")	returns TRUE
MATCHES("abc123", "ab*")	returns TRUE
MATCHES("abc", "{a-f}{qbn}c")	returns TRUE

A.6 Date Boolean Tests

Date values can be compared using '<', '>' and '=' or '==' and '!=' and '~='.

The date times compared are precise to a millisecond.

[date1] < [date2] is true if date1 occurs before date2

[date1] > [date2] is true if date1 occurs after date2

[date1] = [date2] is true if date1 is at the same time as date2

For example:

[date1] < [date2]

NOTE: This is identical to doing: [date1.msec] < [date2.msec]

A.7 Date and Time Functions

TIME( ) Returns the current time as a string - hours:minutes:seconds.

This can be used as a string or assigned to a date variable which will set it to the current day at the time specified. For more formatting options, set the Date variable to NOW() and use the date variable formatting methods.

Returns: String Arguments: None

Example:

[S] = “Started at: “ + TIME()

DATE( ) Returns the current date as a string - month, may, year.

This can be used as a string or assigned to a date variable. (The exact formatting is dependent on the localizations settings of the computer.) See other formats for the DATE(...) function below.

Returns: String Arguments: None

Example:

[S] = “Today is: “ + DATE()

NOW( ) Returns the current date and time as a string - month, day, year hours:minutes:seconds

This can be used as a string or assigned to a date variable. (The exact formatting is dependent on the localizations settings of the computer.)

Returns: String Arguments: None

Example:

[S] = “Today is: “ + NOW()

NOWMSEC( ) Returns the current date and time as a numeric value of the milliseconds since Jan 1, 1970.

This can be used as a numeric or assigned to a date variable.

Returns: String Arguments: None

This can be used for timers, such as setting the NOWMSEC() when the system starts and subtracting it from NOWMSEC() at the end to know how long the system took in milliseconds. A similar technique can be used to see how long it takes a user to answer particular questions.

Example:

[X] = NOWMSEC()

DATE(str)

DATE(Year, Mth, Day)

DATE(Year, Mth, Day, Hour, Min, Sec)

DATE(Year, Mth, Day, Hour, Min, Sec, Msec)

Date(…) can be used to create a date value in an expression without having to create a date variable.

Date(str) can be used for a date where s is a string representation of the date. If the string is empty DATE(), the date is the current day and time.

DATE(Year, Mth, Day) can be used to create a date from the numeric values of year, month and day values.

DATE(Year, Mth, Day, Hour, Min, Sec) can be used to create a date from the numeric values of year, month, day, hour, minute and second values.

DATE(Year, Mth, Day, Hour, Min, Sec, Msec) can be used to create a date from the numeric values of year, month, day, hour, minute, second and millisecond values.

Returns: Date value Arguments: See below

The syntax of the string expression parameter can be any of the formats in the Regional Settings.

Examples:

DATE("March 1, 2011”)	creates the date "March 1, 2011"
DATE("3/1/2011")	creates the date "March 1, 2011"
DATE( )	creates a Date for the current date and time
DATE(2011, 3, 1)	creates the date "March 1, 2011” numeric components of the date: year, month, day

DATE(2011, 3, 1, 11, 10, 07) creates the date "March 1, 2011 11:10:07"

DATE(2005, 3, 1, 11, 00, 00, 1) creates the date 1 millisecond after "March 1, 2005 11:00:00"

DAYSDIFF(date1, date2)

Returns the absolute integer number of calendar days between two dates.

The order of the dates is not important and the function will always return a positive number. The number of days is always rounded up. If the date interval includes any portion of a day, that day is counted. Two dates on the same day will return 0.

Returns: Numeric Arguments: Dates

Examples:

DAYSDIFF([today], [tomorrow]) = 1

DAYSDIFF([one_sec_before_midnight], [one_sec_after_the_same_midnight]) = 1
because the calendar day starts at midnight.

DAYSDIFF([now], [5_minutes_from_now]) = 0
until the time is within 5 minutes of midnight, at which time it will return 1 since the time period would be over 2 different days.

CREATEDATE(date, y_offset, m_offset, d_offset)

CREATEDATE will return a new date based on a starting date and an offset of years, months and days. The offsets can be positive or negative.

Returns: Date string Arguments: Date and numerics

date	The starting date
y_offset	The number of years to add
m_offset	The number of months to add
d_offset	The number of days to add

Each of the offsets must be an integer, but can be 0 or negative.

Examples:

CREATEDATE([today], 0, 0, 1)	will return tomorrow
CREATEDATE([today], 0, 0, 45)	will return a date 45 days from now
CREATEDATE([today], 1, -1, 0)	will return the date 11 calendar months from now because 1 year - 1 month = 11 months

A.8 File Functions

EXISTS(filename)

Returns TRUE if the file exists, otherwise FALSE. This can be used to check to see if a file exists before trying to read it.

Returns: Boolean (False=0, True=1) Argument: String

Example:

IF (Exists(“data.txt”))

A.9 Constants

Corvid recognizes several constants that can be used in expressions.

TRUE	Numeric 1
FALSE	Numeric 0
PI	3.14159..
TAB	The tab character as a string.
CR	The carriage return character as a string.
LF	The line feed character as a string.
CRLF	A carriage return Line feed as a string.

Appendix B - Variable Properties

B.1 What are Properties

All variables in a system have a value. However, they also have other properties, which provide additional information about the value or provide ways to display the variable in various formats. These other properties are obtained by using the notation:

[varname.property]

The property may be a single word, or a word plus a modifier. The properties provide other ways to display or format the value. A variable property is “read only”. Only the variable’s value can be assigned - properties are just a way to get additional information on the value assigned.

Assignments always use just the variable name:

[varname] = ...

In expressions, just the variable name can also be used:

[X] > [Y]

and, since it is an expression, Corvid will replace the variables with their values.

However in reports (and other cases where text is expected), [X] will be replaced by the variable’s **Prompt plus the value** to make it more readable. Corvid is good at guessing how to replace [varname] based on the context, but sometimes a different format is desired. In these cases, the Property gives you complete control on what will be used.

Embedded in text with [[ ]] - Any variable and property can be embedded in any string using double square brackets. Any place [[var]] can be used, [[var.property]] can be used.

B.2 TIME and AGE Properties

The TIME and AGE properties apply to all variables regardless of Type.

These are used to document when a value was set and measure how old the data is.

.TIME - Time the value was set

The property .TIME will return a string stating exactly when the value for the variable was last set or changed. This value is updated each time input or logic sets / changes the value. This can be useful in documenting when a system was run or determining how long it takes end users to answer questions. The string returned can be assigned to a Date variable to use it in expressions and data calculations.

.AGE - Milliseconds since the value was set

The property .AGE will return the number of milliseconds since the value of the variable was set or last changed. This is a numeric value. This can be used to keep track of how long it takes to run a system.

[Seconds_To_Run] = ([Last_Question.AGE] - [First_Question.AGE]) / 1000

Property	Sample Value
[PRICE.TIME]	Fri Sep 08 14:37;20 MST 2000
[PRICE.AGE]	5000

B.3 List Properties

Lists are variables with a fixed set of possible values. Each value has a short text and a full text, which may be different. For the examples below, there is a List variable:

Name = [COLOR]

Prompt = The color of the light is

Values:

Red

Blue

Green

No Property

When used in a report with no property specified, the text output is the full text of the prompt followed by the full text of any values set, connected with “AND”. This is the default text and is equivalent to using the .FULL property.

Example:

[COLOR]

would output:

The color of the light is Red

When used in an expression, List variables are a little different from other types. Normally, IF nodes will be just “Variable name = value” without square brackets. This is to make the Logic Blocks easier to read.

Color = Red

However, when combined with other variables in more complex boolean expressions in the “Expressions” tab, the syntax using properties required is [variable name.value] = “value text” where *value text* is the text of the value.

(([COLOR.value] = “Red”) & ([X] > 0))

.FULL - Prompt and Full Text of all values set

The property .FULL is the full text of the prompt followed by the full text of any values set connected with “AND”.

.VALUE - Text of all values set

The property .VALUE is the text for each value set. If there are multiple values set, the values will be separated by a comma and space

.PROMPT - Prompt Text

The property .PROMPT is the Prompt text for the variable

.NUM - Number of first value set

The property .NUM is the number of the FIRST value set. If there are multiple values set, it will still only return the number of the first one in the value list - that is the one with the lowest number. This should usually only be used for List variables that will only have a single value set. The first value is “1”, second is “2”, etc.

.COUNT - number of values set

The property .COUNT is the count of the number of values in the value list that were set.

.CHECK # or .CHECK value - True if value is set

The property .CHECK followed by a number returns a "1" (TRUE) if that value is set and "0" (FALSE) if it is not. This can be used in complex boolean expressions.

Note: that there is a space between the "CHECK" and the value number.

Property	Sample Value
[COLOR.FULL]	The color of the light is Red AND BLUE
[COLOR.VALUE]	Red, Blue
[COLOR.PROMPT]	The color of the light is
[COLOR.NUM]	1
[COLOR.COUNT]	2
[COLOR.CHECK 2]	0 (FALSE)

B.4 Numeric, String, Date and Confidence Variable Properties

No Property

When used in a report with no property specified, the text output is the full text of the prompt followed by the full text of the value. This is the default text and is equivalent to using the .FULL property.

Example:

[PRICE]

would output:

The price of the item is 123.45

When used in an expression, [varname] is replaced by the **value** of the variable. If the variable is a numeric or Confidence variable, it will be a numeric value and should be used in a location where a numeric would be legal syntax. If the variable is a string or date variable, the value will be a string and should be used in a location where a string would be legal syntax. Date variables can also be used in a numeric context, in which case the value will be the date converted to the number of milliseconds since Jan 1, 1970.

[PRICE] > 100

[NAME] = "Fred"

.FULL - Prompt and Full Text of all values set

The property .FULL is the full text of the prompt followed by the full text of any values set connected with "AND".

.VALUE - Text of all values set

The property .VALUE is the text for each value set. If there are multiple values set, the values will be separated by a comma and space

.PROMPT - Prompt Text

The property .PROMPT is the Prompt text for the variable

.FORMAT fmtStr - Formatted output of value

The property FORMAT allows a numeric or confidence or date variable's value to be formatted to control the number of digits to the right of the decimal, leading, and trailing 0's, etc. Date variables can be formatted to control the precision of the date. The .FORMAT property has no meaning for String variables.

Numeric and Confidence Variables

The format is controlled by a format string following the .FORMAT in the square brackets. The format string specifies the format and follows the standard Java Decimal Format syntax:

Character	Meaning
0	A digit.
#	A digit, but don't show leading or trailing spaces.
.	Location of decimal separator.
,	Location of grouping separator.
;	Separates formats for positive and negative numbers (Positive numbers will use the format left of the ; and negative numbers will use the format right of the ;
-	Negative prefix.
%	Multiply by 100 and shows as percent.
Other Char	Echo in output string.

NOTE: if the formatted value is to be used as a numeric, it must be only numbers, plus, minus and period. Otherwise it will be a string value.

For example: If the value of [PRICE] is 123.45:

[PRICE.FORMAT ###]	would output	123
[PRICE.FORMAT 0000.##]	would output	0123.45
[PRICE.FORMAT $###.##]	would output	$123.45
[PRICE.FORMAT $###.##;($###.##)]	would output	$123.45
but if the value were -555.23,	it would output	($555.23)

Special Format Properties for Date Variables

Date variables have special output format properties:

[D.FORMAT fmtStr]

will output the date formatted by the format string.

The meaning of the fmtStr is dependent on the system localization. The options are:

DATE_SHORT	The shortest form of a date (e.g. 4/5/01)
DATE_MEDIUM	A longer form (e.g. 05-Apr-01)
DATE_LONG	A long form of the date (e.g. April 5, 2001)
DATE_FULL	The longest form of the date (e.g. Thursday, April 5, 2001)
TIME_SHORT	The shortest form of a time (e.g. 2:20pm)
TIME_MEDIUM	A longer form (e.g. 2:20:34pm)
TIME_LONG	A long form of the time (e.g. 2:20:34PM MDT)
TIME_FULL	The longest time format (e.g. 2:20:34 o'clock PM MDT)

The fmtStr can combine any DATE_* with any TIME_* or just a DATE_* or just a TIME_*. If both DATE and TIME are specified, they should be separated by a space in the fmtStr.

For example:

[D.FORMAT DATE_MEDIUM TIME_SHORT]

would output a medium format date string with a short format time string.

[D.PFORMAT fmtStr]

will output the date formatted by the format string, but preceded by the prompt of the variable.

.PFORMAT fmtStr - Formatted output with Prompt

The property PFORMAT is the same as .FORMAT, but the value is preceded by the prompt.

[PRICE.PFORMAT $###.##]

would output

The price of the item is $123.45

.MSEC - Convert Date to Milliseconds (Date Variables Only)

The property MSEC returns the number of milliseconds since Jan 1, 1970. This can be used to do calculations of time between dates stored in Date variables. This is applicable only to Date variables.

.DOW - Convert Date Variable to Day of Week (Date Variables Only)

The property DOW returns a number corresponding to the day of the week. Sunday=1, Monday=2…Saturday=7. This can be used in logic that needs to know the day of week based on a date. his is applicable only to Date variables

For example: [D.DOW] = 2

B.5 Collection Variable Properties

A Collection Variable is a variable whose value is a list of strings. These value strings can be assigned in various ways. The individual strings in the list can have an optional sort value the determines where the string is in the list - the higher the sort value, the higher the string appears in the list. (Closer to the top of the list)

For the examples below:

[DOG] is a Collection variable with prompt *The best breed of dog for you is* and a list of values set by the rules:

Beagle

Labrador Retriever

Golden Retriever

No Property

If there is no property specified, the text output is the full text of the prompt followed by the values concatenated together with a space between them. This is the default text and is equivalent to using the .FULL property with no separator text.

Example:

[DOG]

would output:

The best breed of dog for you is Beagle Labrador Retriever Golden Retriever

.FULL Separator - Prompt and values with optional separator string

The property .FULL is the prompt followed by the values. If the optional separator string is added, it will be added between each value. The separator word will be padded on each side with a space. If the separator word in not included, the values will be separated by a space.

[DOG.FULL OR]

would output:

The best breed of dog for you is Beagle OR Labrador Retriever OR Golden Retriever

.VALUE Separator - All values, with a separator string

The property .VALUE is the text of the value(s). If the optional separator text is added, this will be included between values. The separator word will be padded on each side with a space. If the separator word in not included, the values will be separated by a space.
(Also see .CONCAT below)

[DOG.VALUE]

would output:

Beagle Labrador Retriever Golden Retriever

[DOG.VALUE OR]

would output:

Beagle OR Labrador Retriever OR Golden Retriever

.COUNT - Number of values

The property .COUNT is the number of values in the value list.

[DOG.COUNT]

would output:

3

.FIRST - First item in list

The property .FIRST is the first string in the value list.

[DOG.FIRST]

would output:

Beagle

.LAST - Last item in list

The property .LAST is the last string in the value list.

[DOG.LAST]

would output:

Golden Retriever

.ITEM # - Item # as a string

The property .ITEM # is the value string number # in the value list. The first item in the list is 1. If the list does not have # items in it, the text output will be an empty string.

[DOG.ITEM 2]

would output:

Labrador Retriever

.TOP # - Top # items

The property .TOP # is the top # items from the list of values. The values will be separated by a space. This is most useful when the values in the Collection Variable were sorted since it will return the top # that had the highest sort values.

[DOG.TOP 2]

would output:

Beagle Labrador Retriever

.SCORE # - Sort value for # item

The property .SCORE # will return the sort value for the # item in the list. The values must have been added with sort for this to be used.

[DOG.SCORE 2]

would output:

7

.CONCAT Separator - Concatenate to a string

The property .CONCAT is the list of values concatenated into a single string.

The syntax is [coll.CONCAT str] where str is an optional separator string, which will be included between values. Unlike the similar .VALUE property, the separator string will **NOT** be padded on each side with a space if it is put in quotes, providing more control. If the separator string is not included, the values will be separated by a space.

[DOG.CONCAT AND]

would output:

Beagle AND Labrador Retriever AND Golden Retriever

The optional separator string can include the following line control characters:

\n	New Line
\r	Carriage Return
\t	Tab

If the separator string is in quotes, the exact quoted string (without the quotes) will be used to separate values. If the string is not in quotes, the values will be separated by a space, str, and a space.

For example, if there is a Collection variable [coll], with values "aaa", "bbb", "ccc" and no prompt text:

[Coll]	aaa bbb ccc
[coll.concat AND]	aaa AND bbb AND ccc
[coll.concat "AND"]	aaaANDbbbANDccc (Only quoted string added without spaces)
[coll.concat "\r\n"]	aaa bbb ccc (one value on each line)

.INCLUDES *text* - TRUE if *text* is in the list

The property .INCLUDE *text* will be “1” (TRUE) if the text matches any value in the variable’s value list and “0” (FALSE) if it does not. The entire text must match, but it is not case sensitive.

Properties that return True or False can be used in IF nodes or other Boolean tests. They can also be used in numeric expressions.

[DOG.INCLUDES Beagle]

would output:

1

[DOG.INCLUDES Poodle]

would output:

0

INCLUDES can accept an embedded variable as the value to test against.

For example:

[THINGS.INCLUDES [[test_str.value]]]

where [THINGS] is a Collection variable, and [test_str] is a string to test to see if it is one of the items in the collection.

.NOTINCL text - TRUE if text is NOT in the list

The property .NOTINCL *text* is the opposite of .INCLUDES. It will output “0” (FALSE) if the text matches any value in the variable’s value list and “1” (TRUE) if it does not.

[DOG.NOTINCL Beagle]

would output:

0

[DOG.NOTINCL Poodle]

would output:

1

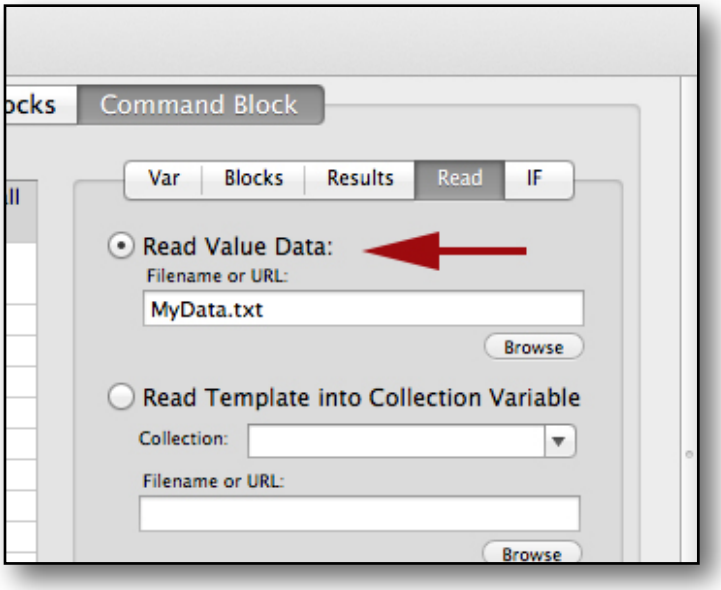
Appendix C - Reading Data from External Sources

C.1 Specifying the File to Read

Corvid can read a file of data.

This is done by selecting the “File” in the Command Block Builder “Read” tab.

Enter the name of the file to read, or browse to it. The location specified can just be the name of the file if it is in the same folder as the other system file or a URL to any place on the web. When a URL is used to a data not local to the system files, a full URL (http://www....) can be used.



C.2 Calling External Programs

Since a URL can be used for the “file”, it can be a URL to a servlet or other active program that takes parameters and returns data as a file. For example, a servlet could be called:

http://myServer.com/myServlet/myClass?ID=[[ID_Var.value]]

In this case the value of the Corvid variable [ID_Var] would be embedded in the servlet call and passed to the called servlet. This could pass data to the servlet for it to use or to identify the data to find / calculate and return.

The called program can be anything provided:

a. It can be called by a URL. This means it must be a Java Servlet or other type of program designed to run via the web

b. It must return data in the form expected by Corvid

C.3 Format of Returned Data

The returned data MUST just be text. It should not have a header that will be interpreted as part of the data. An HTML page should not be used because it will have various tags wrapped around the text of the data.

To create static files of data, use a program such as Notepad that produces simple text files. If you use a word processor unless you make sure to save the file as “text”.

The returned data should be one or more name / value pairs that identify a variable followed by the data to assign to that variable.

[varname] value

The variable identifier is the name of the variable in square brackets, followed by a space and the value to assign to that variable.

The value is just a text string, but must match the variable's type. A numeric variable must be assigned a numeric value. A string can be assigned any value.

There can be multiple name / value pairs, one per line.

For example:

[X] 123

[Temp] 77

[Price] 99.99

would set the values of the 3 variables [X], [Temp] and [Price].

The format of the data returned for a variable depends on the type of variable.

List Variables

The value assigned to a List variable can be:

The number of the value. (First value = 1, second = 2, etc)

The text of the value

More than one value number or text separated by the TAB character

A numeric expression that evaluates to the number of a value

Example: List variable [Color] with values "Red", 'Blue" and "Green"

Returning	**Sets**
[Color] 1	Red
[Color] Blue	Blue
2 3 (separated by Tab)	Blue, Green
[Color] Red Green (separated by Tab)	Red, Green
Color] 1+1	Blue

Numeric Variables

The value assigned to a Numeric variable can be:

A numeric value.

An expression that evaluates to a numeric value.

Example: Numeric variable [Price]

Returning	**Sets**
[Price] 99.95	99.95
[Price] 90 + 10	100

String Variables

The value assigned to a String variable must be a string value without quotes

Returning	**Sets**
[Name] Exsys	Exsys

Date Variables

The value assigned to a Date variable must be string date value. This can be a full date “July 5, 2010” or shorter forms such as “1/5/99” or the number of milliseconds since Jan 1, 1970. Dates can also include the time.

Remember: All date values are interpreted by the local settings for dates on the machine running Corvid. This is the server settings for the Servlet Runtime and the client machine settings for the Corvid Applet Runtime. Different countries use different formats for dates. Dates such as “1/2/10” can be ambiguous since the 1 is the month in some countries and the 2 is the month in others. When assigning dates, the long form (e.g. July 5, 2010 1:23PM) is best to use since it is unambiguous. To set the variable to the current date, “now()” can be used, but this can also be done in Corvid and does not require an external call. Using "now()" with the Servlet Runtime will set the time for the server. The Applet Runtime will set the time for the client PC.

Example: Date variable [Start_date]

Returning	**Sets**
[Start_date] 5/12/10	May 12, 2010 (in the US)
now()	The current date and time
Aug 21, 2008 4:52PM	Aug 21, 2008 4:52PM

Confidence Variables

The value assigned to a List variable can be:

A numeric value consistent with the confidence mode.

An expression that evaluates to a value consistent with the confidence mode.

Remember: The value assigned must be consistent with the confidence mode settings of the variable. A variable that expects a value between 0 and 1, must be given a value in that range. Confidence values assigned will be combined with any other values assigned to the variable based on the confidence mode settings.

Example: Confidence variable [Conf]

Returning	**Sets**
[Conf] .25	.25 combined with any other values
[Conf] 5 + 10	15 combined with any other values

Collection Variables

The value assigned to a Collection variable must be a string value without quotes.

Multiple string values separated by the TAB character

Multiple lines in the data file for the same Collection variable will add multiple items.

Items added to a Collection variable are always added to the end of the list. To add items to a collection using the other methods such as ADDFIRST or sort, read the returned data into a string variable and then add the value to the collection using a SET command with a method in the Command file.

Example: Collection variable [Coll

Returning	**Sets**
[Coll] aaa bbb (separated by Tab)	Adds “aaa” and “bbb” to the list
[Coll] aaa	
[Coll] bbb	
[Coll] ccc (each on a separate line)	Adds “aaa” “bbb” and “ccc” to the list.

Index