Symbolic Ai

User's Guide

Version 2.2

December 9, 2021

ILS Automation, Inc.

Table Of Contents

Table of Contents

1 Introduction 7

2 Application / Family / Diagram / Final Diagnosis Overview 9

2.1 Application 9

2.2 Family 10

2.3 Problem / Final Diagnosis 10

3 Symbolic Ai Features 10

3.1 Application Configuration 10

3.2 Output constraints 12

3.3 Minimum Change Processing 13

3.4 Problem Prioritization 14

3.5 Multiple Problem Handling 14

3.6 Calculation Method 14

3.7 Text Recommendation 15

3.8 Manual Move 15

3.9 Constant Final Diagnosis 18

3.10 Input/Output and Source/Sink Blocks 18

3.10.1 Important Features 19

3.10.2 Creation Shortcut 20

3.11 Diagnosis Queue / Problem Explanations 21

4 Symbolic Ai Blocks 22

4.1 Palette Organization 23

4.2 Block Behavior 23

4.3 Block Definitions 23

4.4 Block Descriptions 29

4.4.1 Action - Misc 29

4.4.2 And - Logic 29

4.4.3 Arithmetic - Arithmetic 30

4.4.4 Average – Statistics 30

4.4.5 Bias - Arithmetic 32
4.4.6 Change Sign - Arithmetic 32
4.4.7 Command - Control 32
4.4.8 Compare Blocks - Observation 33
4.4.9 Conditioner - Connectivity 36
4.4.10 Data Selector - Control 37
4.4.11 Data Shift - Control 37
4.4.12 Data Switch - Control 38
4.4.13 Delay – Times & Counters 38
4.4.14 Difference - Arithmetic 39
4.4.15 Discrete Rate of Change - Arithmetic 39
4.4.16 Edge Trigger - Control 40
4.4.17 Equality - Observation 40
4.4.18 Explanation - Conclusion 41
4.4.19 Exponential - Arithmetic 42
4.4.20 Filter - Statistics 42
4.4.21 Final Diagnosis - Conclusion 42
4.4.22 Gain - Arithmetic 45
4.4.23 High Limit - Statistics 45
4.4.23 High Limit Blocks - Observation 45
4.4.24 High Selector - Statistics 49
4.4.25 High Value Pattern - Observation 50
4.4.26 In Range - Observation 51
4.4.27 Inference Memory - Control 52
4.4.27 Inhibitor - Control 53
4.4.28 Input - Connectivity 56
4.4.29 Inverse - Arithmetic 57
4.4.30 Junction - Connectivity 57
4.4.31 Lab Data - Connectivity 57
4.4.32 Linear Fit - Arithmetic 58
4.4.33 Ln - Arithmetic 58

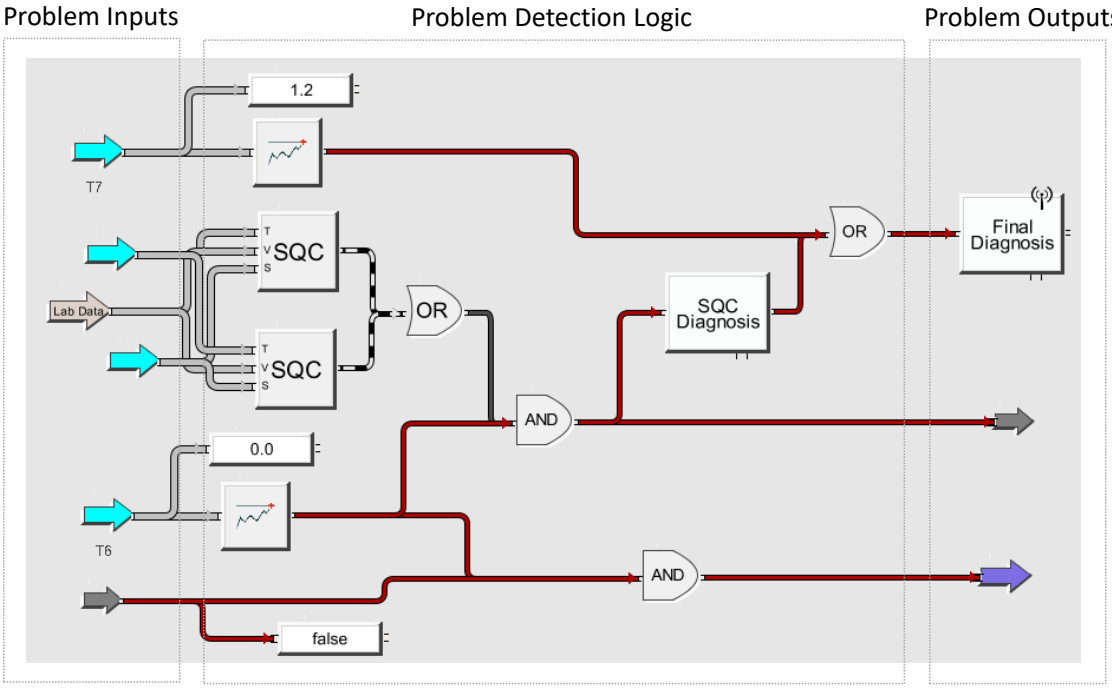
4.4.34 Log10 - Arithmetic....59
4.4.35 Logic Filter - Logic....59
4.4.36 Logic Latch - Logic....60
4.4.37 Low Limit - Statistics....60
4.4.38 Low Limit Blocks – Observation....61
4.4.39 Low Selector - Statistics....63
4.4.40 Low Value Pattern - Observation....63
4.4.41 Median - Arithmetic....64
4.4.42 N True - Logic....64
4.4.43 Not - Logic....64
4.4.44 Note - Misc....65
4.4.45 Or - Logic....65
4.4.46 Output – Connectivity....66
4.4.47 Parameter – Connection....66
4.4.48 Persistence Gate – Timer & Counters....66
4.4.49 Product - Arithmetic....67
4.4.50 Set Property - Control....67
4.4.51 Qualified Value - Misc....68
4.4.52 Quotient - Arithmetic....69
4.4.53 Random - Statistics....69
4.4.54 Range - Observation....70
4.4.55 Readout - Misc....70
4.4.56 Receiver - Connectivity....71
4.4.57 Reset - Control....71
4.4.58 Sink – Connectivity....71
4.4.59 Source - Connectivity....72
4.4.60 SQC - Statistics....72
4.4.61 SQC Diagnosis - Conclusion....74
4.4.62 State Lookup - Control....74
4.4.63 Statistics – Statistics....75
4.4.64 Sub Diagnosis - Conclusion....76

4.4.65 Time Readout - Misc 77
4.4.66 Time Fork - Misc 77
4.4.67 Timer – Timers & Counters 77
4.4.68 Trend Detector - Statistics 78
4.4.69 Transmitter - Connectivity 79
4.4.70 Truth Counter – Timers & Counters 80
4.4.71 Truth Pulse - Connectivity 80
4.4.72 Unknown - Logic 80
4.4.73 XY Fit - Arithmetic 80
4.4.74 Zero Crossing - Observation 81
4.5 Bad Data Handling 81
4.6 Block Locking 81
4.7 Block Reset 82
4.8 Timestamp Handling 82
4.9 Connection Type Change 82
4.10 Type Coercion 83
4.11 Creating Custom Bocks 83
4.11.1 Block Icons 84
4.11.2 Init File Requirement 84
4.11.3 Block Properties 84
4.11.4 Sample User Block 88
4.12 Signal Processing 90
5 Client User Interface 91
5.1 Operator Console 92
5.2 Notification of New Recommendation 93
5.3 Setpoint Spreadsheet 93
5.3.1 Absolute Recommendations 95
5.3.2 Recalculating Recommendations 95
5.3.3 Output Clamping 95
6 Advanced Topics 100
6.1 Final Diagnosis Processing 100

6.2 Data Consistency 101
6.3 Download Processing 102
6.4 Triggering a Final Diagnosis Externally 102
6.4.1 Triggering a Final Diagnosis from a Window 102
6.4.2 Triggering a Final Diagnosis from an SFC 103
7 Trouble Shooting 105
7.1 Diagram Related Problems 105
7.2 Recommendation Related Problems 105
7.3 Logbook and Queue Messages 106
7.3.1 Logbook Message Details 107
8 Database Tables 108
9 Programmer's Interface 108

1 Introduction

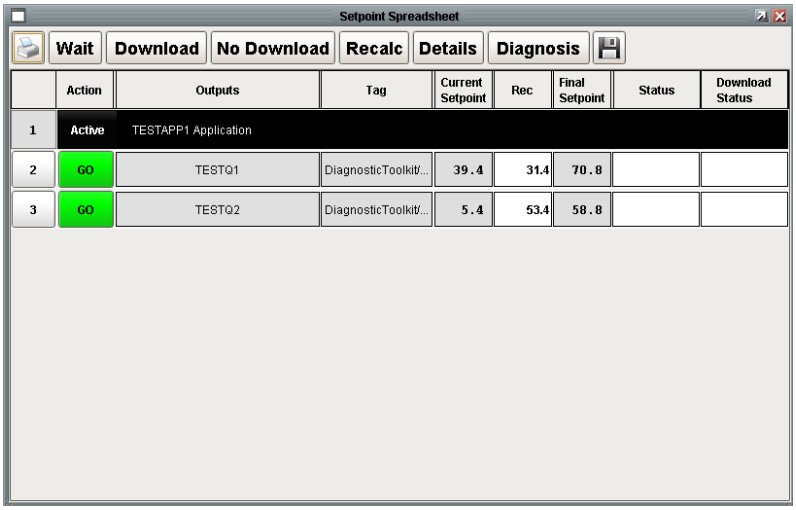
The Diagnostic Toolkit provides facilities for diagnosing and correcting problems in a real-time closed-loop manner. An application is designed by identifying families of problems. An application generally consists of several families. Each family consists of one or more problems to be diagnosed and corrected. A problem consists of a graphical diagnostic diagram. A typical diagram is shown below:



A diagram flows from left to right. The left side of the diagram has input blocks for the problem's inputs. The right side of the diagram has the problem's outputs which may include any number of Final Diagnosis and / or outputs. The outputs go to tags which are used either by other diagnostic diagrams or other toolkit facilities. The middle portion of the schematic implements the detection logic and contains various blocks that transform, condition, and test data. Blocks use connections to transmit data from one block to another. Numeric data is transferred over thick grey connections. The output from an observation and the input to logic blocks and diagnosis are Boolean values. Connections that carry Boolean values are animated to communicate their value. A red connection is False, a green connection is True, Grey is unknown, and dashed is no value.

The final diagnosis block has a very important role in the Diagnostic Toolkit. When a final diagnosis becomes true it is a signal that action needs to be taken to maintain peak performance in the plant. There are a set of outputs that are associated with a final diagnosis that are determined by the process expert and are the control variables that affect

the critical performance criteria being monitored. The final diagnosis also contains a reference to a calculation method. A calculation method is a Python script written by the process expert. It returns a list of numeric recommendations for changes to be made to the control variables to correct the problem move the state of the plant back to optimal. The recommendations are then handled by the diagnostic toolkit which validates, conditions, combines and finally presents them to the operator for confirmation.



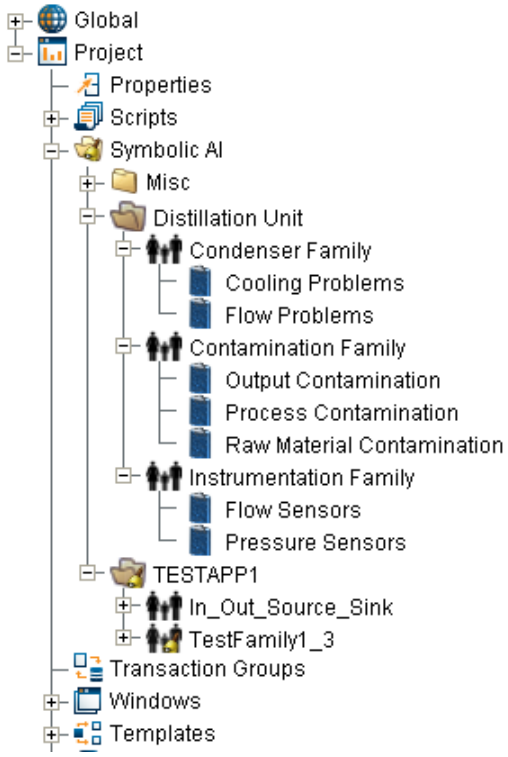
The setpoint spreadsheet shows a row for each output. It shows the tag, the current value, the recommended change amount, and the final setpoint. The operator has numerous options including to accept the recommendations and download them to the control system. The other options available to the operator will be discussed in detail later.

The toolkit also contains the following capabilities, all of which will be described in detail:

- A two level prioritization scheme for ensuring that the most serious problem is worked first.
- A means to combine multiple recommendations when more than one final diagnosis is active.
- Output validation to ensure that recommendations are reasonable and within predefined constraints.
- Awareness of time based data so that data collected before a recent change to the plant is discarded.
- A way for an operator to manually initiate a move subject to the output validation constraints described above.
- Robust logging for troubleshooting

2 Application / Family / Diagram / Final Diagnosis Overview

Symbolic Ai uses a hierarchy of applications, which contain families, which contain diagrams, which contain final diagnosis. A diagram is a graphical schematic with one or more final diagnosis. The hierarchy is built in the designer using the project browser as shown below. The BLT module defines a new resource type which appears in the project tree. Right clicking on the Symbolic Ai node displays a menu choice to create an application. From an application you create a family and from a family a diagram.



2.1 Application

An application is the set of diagnostics that apply to a processing unit / product. Symbolic Ai will process applications independently and clearly present their recommendations on the same setpoint spreadsheet, separated by application. Applications should generally be independent of each other. Configuration of the application includes defining the set of control variables that control the process within that unit. These control variables, hereafter referred to as outputs, will be available to the calculation methods associated with the final diagnosis that belong to the application. An indication that an application is well designed

is that the sets of control variables do not overlap. If the same control parameter were in both application then optimizing one may de-optimize the other.

2.2 Family

A family is a way of organizing and prioritizing diagrams. An application can have many families and a family can have many diagrams. The Symbolic Ai engine will act on the highest priority final diagnosis in the highest priority family.

2.3 Problem / Final Diagnosis

A Final Diagnosis represents a problem. When the input to a final diagnosis becomes *True* it represents the definitive statement about the existence of a problem in the plant. The definition of a final diagnosis requires a complete understanding of the problem including how to correct the problem. The corrective action may take one of three forms:

1. Quantitative: specific numeric recommendations to process control variables. Quantitative recommendations are determined dynamically by invoking a user written calculation method when the problem becomes active.
2. Text: instructions of manual actions to take to correct the problem. A text recommendation is generally configured statically but may be created dynamically by a calculation method.
3. Constant: locks out all lower priority problem, typically used for unit shutdown or changeover situations.

When a final diagnosis becomes true, then a diagnosis entry is made in the application's diagnosis queue. This entry is in turn published to the appropriate console diagnosis queue. If the final diagnosis uses quantitative or text recommendations, and it is determined to be the highest priority problem, then the console operator will be notified of the recommended changes to correct the situation. An operator alert, aka loud window, will be posted to get the operator's attention. When the operator acknowledges the operator alert then either the text recommendation or a spreadsheet showing the quantitative recommendations will be displayed.

The Final Diagnosis block is discussed in more detail in section *4.4.21 Final Diagnosis - Conclusion*

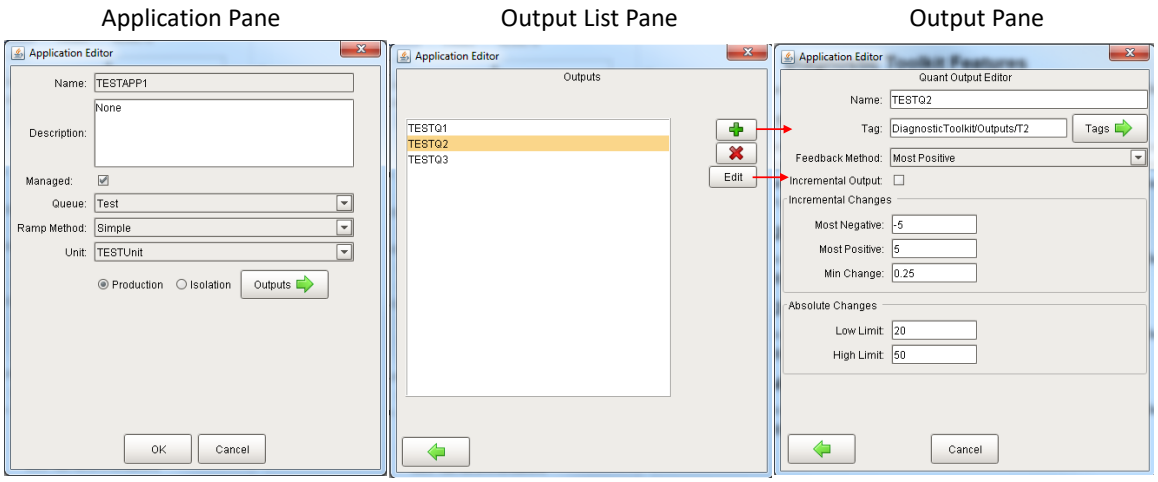
3 Symbolic Ai Features

The diagnostic toolkit provides many powerful features that are described in more detail below.

3.1 Application Configuration

Configuring an application includes defining the set of control variables that control the process. These control variables, hereafter referred to as outputs, will be available to the

calculation methods associated with the final diagnosis that belong to the application. The configuration dialog for an application provides a three pane window.



The first window, shown on the left, define basic properties of the application.

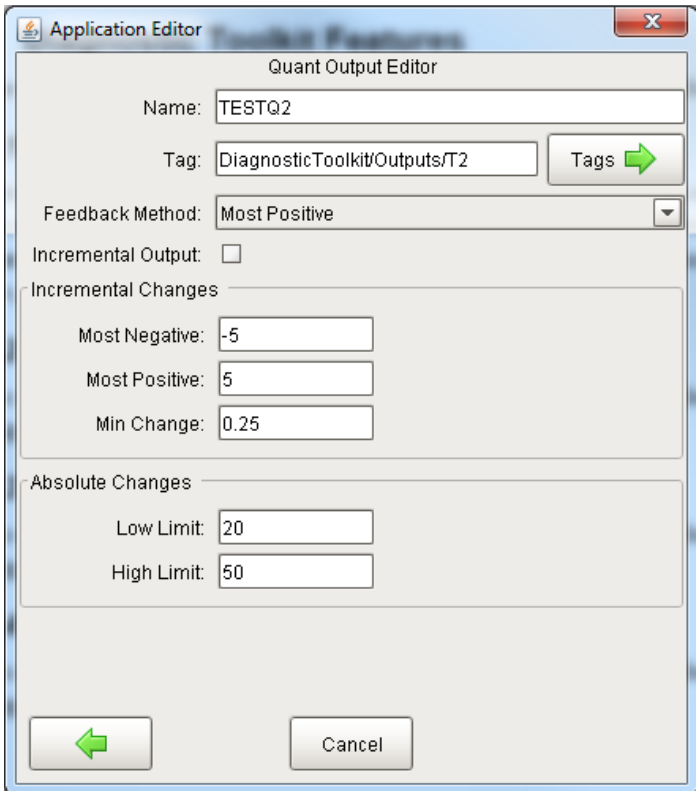
Property	Description
Managed	Species if this application is active. Diagrams will function and propagate data regardless of this setting, but final diagnosis will only make recommendations if managed.
Queue	The name of the message queue where messages related to the diagnosis will be posted. This is useful for debugging and auditing. The "Ramp Method" is current not used.
Ramp Method	Strategy for calculating the ramp time for multiple simultaneous ramp recommendations. Strategies are None, Shortest, Longest, and Average. If None is selected, then the individually calculated ramp times will be used.
Unit	The name of the unit that this application refers to. A project can monitor multiple applications.
Production/Isolation	Specifies which database will be updated.

The middle pane displays the outputs that are available to all of the diagnosis that are part of the application. New outputs can be defined by pressing the green "+" button. An output can be deleted by pressing the red "X" button. An output can be configured by selecting an output and pressing the "Edit" button.

The final pane defines the details of a specific output including how multiple recommendations for the same output are to be combined. Most importantly, it specifies the path to the OPC tag that determines the location in the DCS that will be affected. Output constraints are discussed below.

3.2 Output constraints

The diagnostic toolkit provides a tool for automatically detecting and correcting problems. Because setpoints are calculated automatically it is important that the toolkit provide a framework that guarantees that the calculated setpoint changes are rational and subject to the guidelines defined for each individual output at the application level. The application of the constraints is discussed in detail in section 5.3.3.



The properties for an output are:

Property	Description
Feedback Method	Specifies how multiple recommendations for the same output are combined. The choices are: Average, Most Negative, Most Positive, Simple Sum.
Incremental Output	Specifies if the values returned by the calculation are absolute or incremental.
Incremental Changes – Most Negative	Defines the greatest change that is allowed in the negative direction.
Incremental Changes – Most Positive	Defines the greatest change that is allowed in the positive direction.

Property	Description
Incremental Changes – Min Change	Defines the minimum change amount. If the absolute value of the recommended change is smaller than this limit then the recommendation will be ignored. If all of the recommendations for a problem are below this threshold then the diagnosis will be cleared by performing an automatic No-Download.
Absolute Changes – Low Limit	Defines the absolute minimal value for an output. This applies regardless of whether the recommendation returns absolute or incremental values. These limits are checked after multiple recommendations are combined.
Absolute Changes – Low Limit	Defines the absolute maximum value for an output. This applies regardless of whether the recommendation returns absolute or incremental values. These limits are checked after multiple recommendations are combined.

3.3 Minimum Change Processing

Minimum Change Processing is concerned with how small recommendations are handled. There isn't a one size fits all answer. In many situations it is desirable to hide insignificant changes from the operator. For example, a change of 0.001 in the feed rate of a main feed is insignificant and should probably not be shown. In other situations, it is desirable to see all recommendations, regardless of magnitude. For example, when a diagnostic problem is initiated by an operator such as in rate change, the operator expects the setpoint spreadsheet to be displayed, regardless of how small the changes are. It is common when testing to specify a very minor change in the rate which would produce correspondingly small recommendations.

There are several settings that affect minimum change processing:

- Quant Output defined on the application property editor:
    - Minimim Increment: This defines the magnitude of a recommendation that is deemed insignificant.
    - Ignore Minimum Increment: This is a dynamic property, it is reset to False at the beginning of every management cycle, it must be set to True in the calculation method.
- Final Diagnosis property editor:
    - Trap Insignificant Recommendations.

After the final recommendations are calculated, the recommendation will be suppressed if *TrapInsignificantRecommendation* is True and *IgnoreMinimumIncrement* is False. Because this happens after recommendations are combined and there could be recommendations from multiple final diagnosis for a final diagnosis. This is considered in

mergeOutputs() in finalDiagnosis.py after recommendations from multiple final diagnosis are combined. If multiple final diagnosis are touching the same output and the final diagnosis have different values for *TrapInsignificantRecommendation*. The value will be calculated by taking the OR of the values.

3.4 Problem Prioritization

The diagnostic toolkit prioritizes problems using a two-tier numeric prioritization. A numeric priority is assigned to a family and to a final diagnosis. The priority is a non-negative floating-point number. The lower the number the higher the priority. It is normal to have multiple problems and/or final diagnosis with the same priority. The framework will act on the highest priority final diagnoses from the highest priority families.

3.5 Multiple Problem Handling

The diagnostic toolkit is designed to handle multiple applications and final diagnosis and to combine the recommendations and present them in a unified way to the operator. In a real plant, it is common that when one thing goes wrong, multiple alerts/diagnosis may be triggered. Using the prioritization described above, the most important problems will be acted upon. Because the family/problem priority does not need to be unique, there could be multiple recommendations for the same output. These will be combined based on the feedback method chosen for the output and a single change will be displayed to the operator.

3.6 Calculation Method

The calculation method is written in Python and can be written externally, using the Eclipse IDE or may be in the global (shared) script area. The calculation method takes five arguments and returns three. The five arguments are:

Value	Type	Description
applicationName	String	The name of the application
finalDiagnosisName	String	The name of the final diagnosis
finalDiagnosisId	List of dictionaries	The UUID of the final diagnosis in the Symbolic Ai engine
provider	String	The tag provider
database	String	The database context

The return arguments are:

Value	Type	Description
Success	Boolean	True if the calculation method completed successfully.

explanation	String	A text string that will be appended to the static portion of the recommendation and posted to the application queue.
recommendations	List of dictionaries	A list of dictionaries of the specific numeric recommendations. Each dictionary has exactly two members: "QuantOutput" and "Value". The "QuantOutput" must match one of the quant outputs defined for the final diagnosis.

The objective of the calculation method is to calculate a "move" that will transform the process from its current state to the desired state. The move is generally done by calculating the error which is the difference between desired state and the current state. The error is then used to calculate a setpoint change for each of the control variables defined for the diagnosis. The design of the final diagnosis and calculation method needs to be consistent

A simplified calculation method is shown below.

```
13 def fd1_2_1(applicationName, finalDiagnosisName, finalDiagnosisId, provider, database):
14     print "In fd1_2_1"
15     explanation = "The TESTFD1_2_1 will use data of gain = 1.2, 1.5, and 0.9, SP = 23.4."
16     recommendations = []
17     recommendations.append({"QuantOutput": "TESTQ1", "Value": 31.4})
18     recommendations.append({"QuantOutput": "TESTQ2", "Value": 53.4})
19     return True, explanation, recommendations
```

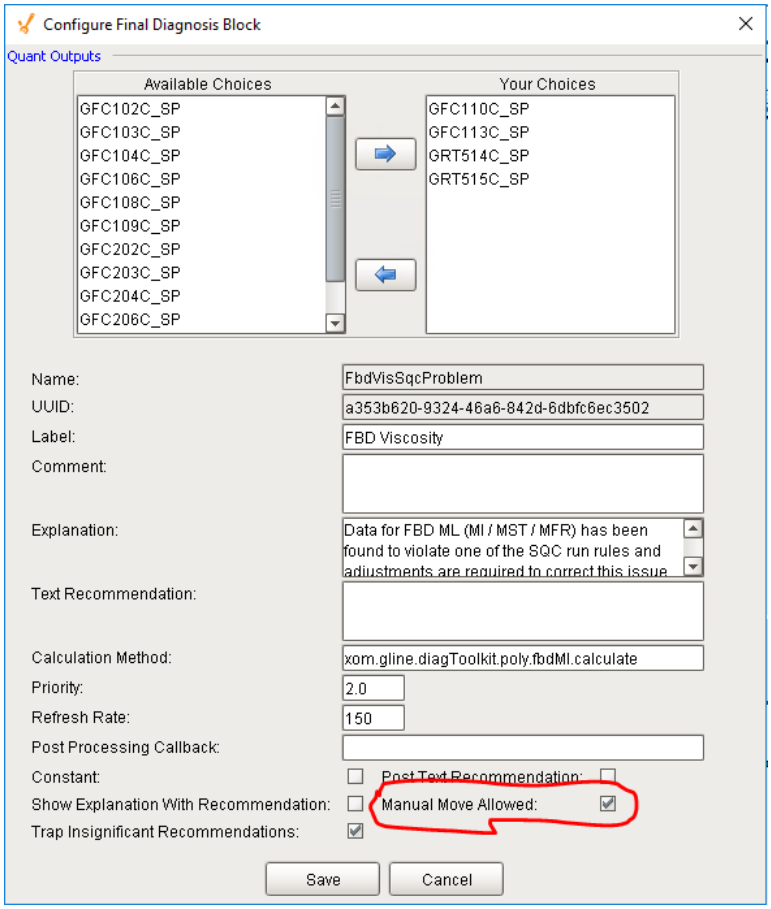
3.7 Text Recommendation

The main strength of the diagnostic toolkit is the ability to diagnosis a problem and make a quantitative recommendation to correct the problem. However, the toolkit also provides the flexibility of making qualitative recommendations in the form of advice. These are called text recommendations. Unlike numeric recommendations which attempt to correct the abnormal event, a text recommendation notifies the operator who must then take action to correct the abnormal event. The advice that is displayed may contain specific instructions tailored to incident and can be statically configured in the configuration dialog for the final diagnosis or it can be constructed dynamically, at the time of the event, via a calculation method callback. Once the text recommendation has been acknowledged by the operator the source diagram will be reset just as if a numeric recommendation were downloaded.

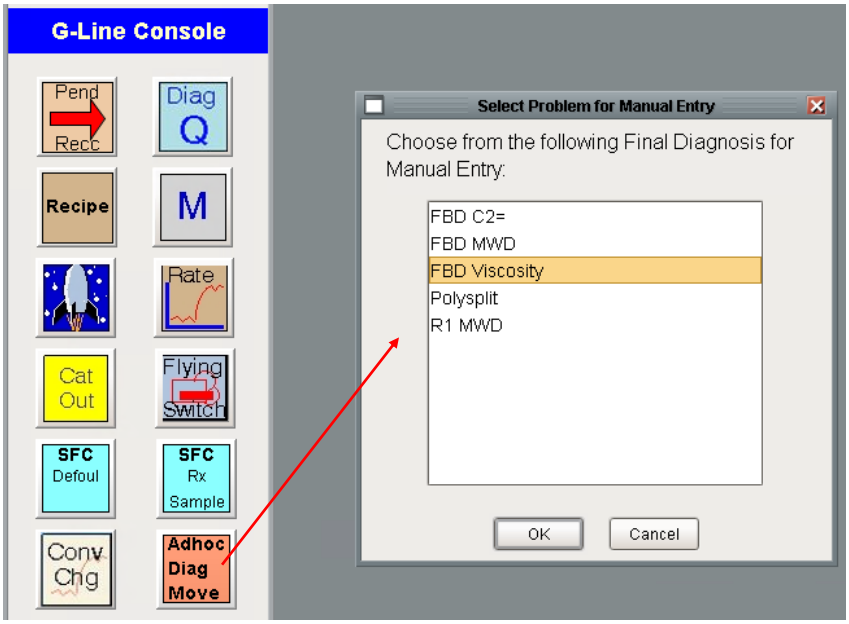
3.8 Manual Move

A Manual Move may be initiated by an operator to implement a change to the system. A manual move may be used if the automated problem detection did not detect a problem due to insufficient problem detection logic, a system data communication problem, a change in product specifications that have not been coordinated with the application, or just process investigation. A manual move is made in terms of a specific problem. For this discussion, the term problem is synonymous with a Final Diagnosis. A problem must be designated

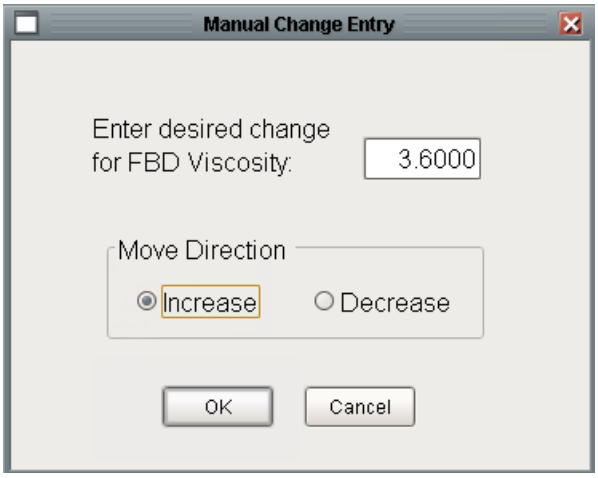
as being appropriate for a manual move. This is done by selecting the Manual Move checkbox on the Final Diagnosis' configuration dialog.



A manual move is initiated by the operator by pressing the "Adhoc Diag Move" button on the operator console. The window shows a list of all of the problems that have been designated to allow manual moves.



The move is entered in terms of an error. The error will be used to calculate the move for each of the outputs defined for the problem. For example, for the FBD Viscosity problem, the problem manipulates four outputs as defined by the configuration dialog shown above.



As mentioned above, the calculation procedure for the problem must be written to use manual moves, as shown below. The calculation procedure should have the following characteristics:

- It should check if the diagnosis is being made naturally or from a manual move initiated by the operator. This is done in line 45 by calling *fetchManualMoveInfoById()*.

- The individual recommendations are calculated in terms of an error (line 63 and 64) which can either be calculated from some current tag values (line 48) or from the manual move system (line 50).
- As described in the comment on lines 54-57, the manual move calculations run in the client that initiated the move. This allows for additional actions such as showing a window (lines 60 and 61).
- If the user presses RECALC on the setpoint spreadsheet that was initiated from a manual move, then this will run in the gateway and it will get into the manual move branch so it is important to check the scope in which it is being called to prevent an attempt to display a window from gateway scope.

```
45  manualMove, manualMoveAllowed = fetchManualMoveInfoById(fdId, database)
46
47  if manualMove in [0.0, None]:
48      error = sp - pv
49  else:
50      error = manualMove
51      log.infof("Using the manual move (%f).", manualMove)
52
53      '''
54      When a manual move is being handled this runs in the client.  Normally, calculation methods run in the gateway.  It runs in the client that
55      initiated the manual move which makes it really easy to show a window. This makes no attempt to show the window on other clients monitoring the post.
56      Note: if the user presses RECALC from the setpoint spreadsheet after initiating a manual move then this will run in the gateway AND we will get inside this
57      branch, so protect against opening a window from gateway scope.  The window should already be open anyway.
58      '''
59      if not isGatewayScope():
60          window = system.nav.openWindow("GLine/Miscellaneous/Steady State Compare Data")
61          positionWindow(window, "LOWER-LEFT", 0.8)
62
63  nr2h2 = gainH2 * error * fr2h2
64  nr2t = gainRxT * error
```

3.9 Constant Final Diagnosis

A "Constant" final diagnosis is generally used to block out other final diagnosis. A constant final diagnosis does not make a recommendation; therefore, it does not specify a calculation method. A constant recommendation is generally used to detect an equipment state and then block out other diagnostic diagrams. A simple example is if the equipment is shutdown or maintenance a common problem of many alerting systems is that they instantly generate all sorts of alerts. A constant final diagnosis with a very high priority that detects that the plant is down is common. This is preferable to adding logic to each schematic to detect normal operating conditions.

3.10 Input/Output and Source/Sink Blocks

There are four blocks that are the main way to bring data into a diagram and to get data out of a diagram. It is important to understand the similarities and differences between the blocks which are highlighted in the following table. All four of the blocks are configured with a tagpath.

Block	Direction	Tag Type	Reset Propagates	Follow Connections
Input	In	OPC, Memory, Expression or Query	No	No
Output	Out	OPC or Memory	No	No
Source	In	Memory tag in connection folder	Yes	Yes
Sink	Out	Memory tag in connection folder	Yes	Yes

Input / Output blocks are intended to bring data into a Symbolic Ai diagram from an external system via an OPC tag or from a derived value via an expression tag or from some other facility in Ignition via a memory tag.

Source / Sink blocks are intended to provide a connection between diagrams. It is a common practice for a diagram to produce a value from a combination of blocks and then that value is pushed to multiple other diagrams. Using memory tags to facilitate the communication between the Sink and the Source provides a convenient debugging capability and still utilizes the well-known tag listener technique. The normal workflow is to create a sink and then create a source and configure it to reference a sink.

3.10.1 Important Features

These blocks have the following important features:

- A blocks connection type is determined by the type of the tag that it is bound to. This is nice automatic capability to make it faster to create and configure a diagram.
- The blocks connection type is "locked" in once it has been connected to another block.
- A block can be reconfigured by dragging a tag onto it. If the block has "locked in" its data type, then the data type of the new tag must be the same as the old tag.
- Blocks can be created by dragging them from the palette or by dragging a tag from the tag browser.
- Creating a block by dragging a tag from the tag browser has the following features:
    - Dragging to the left side of the diagram creates an input.
    - Dragging to the right side of the diagram creates an output.
    - When dragging a tag onto a block, or dragging a tag to create a block, will name the block with the tag name rather than some arbitrary unique name. The name should still be checked for uniqueness on the diagram and be morphed by adding a numeric suffix if necessary.
    - When dragging a tag onto the canvas, the type of the connection is determined by the type of the tag. A string, float, or integer will create an input block with

a Data connection, a Boolean tag will create an input block with a truth value connection.

- Source / Sink Blocks have the following feature. (I think that the normal workflow is to create the sink, name the sink, then create a source, and select the name of the sink. If you want to create the source first you can, but you won't be able to configure it until the sink is created.):
  - Dragging a memory tag from the Symbolic Ai/Connections folder will create a Source / Sink subject to the same left / right rules.
  - Dragging a tag that is NOT in the Symbolic Ai/Connections folder onto a Source / Sink will be rejected.
  - Multiple Sinks cannot reference the same tag
  - Multiple Sources can reference the same tag
  - Source / Sink blocks can only reference memory tags in the Symbolic Ai/Connections folder.
  - Dragging a sink from the palette should uniquely name the sink and create a tag. If the user names the sink it will rename the tag.
  - Dragging a source from the palette will leave the source unnamed. Configuring the Source should be as simple as selecting the name from a list of existing sinks.
  - Deleting a sink should delete the tag. If there are sources referencing the sink, the user will be warned / prevented from deleting the source.
  - Renaming a sink should rename the tag and referenced sources.
  - Manually deleting connection tags will leave any referenced source and sinks in an unusable state.
  - Sources and sinks must support reset propagation, this only applies when the path is a truth-value. (Reset starts with a Final Diagnosis and goes upstream to any observations and resets them. The observations will set their output to UNKNOWN which must then propagate forward.) In order to propagate an UNKNOWN from a sink to a source, truth value connections use string tags. This provides some ambiguity when determining the type of connection for a source because a string tag could be used for a data connection or a truth-value connection. The strategy should be to set the connection type of the source to the same as the connection type of the sink.

3.10.2 Creation Shortcut

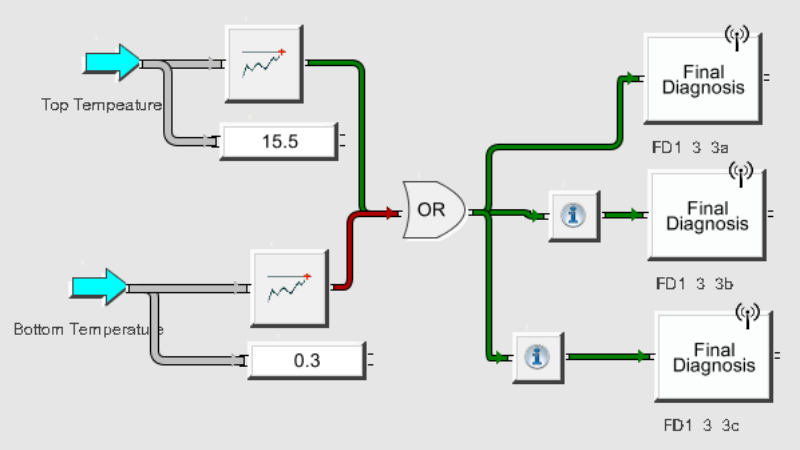
There are two ways to create an Input / Output or Source / Sink block. They can be dragged from the palette onto the designer canvas or tag can be dragged from the tag browser onto

the designer canvas. If the tag is from the Symbolic Ai/Connection folder, then a source / sink will be created any other tag will create an input or output. If the tag is dropped on the left side of the diagram then an input or source will be created, if dropped on the right ride then an output or sink will be created.

3.11 Diagnosis Queue / Problem Explanations

Explanations are dynamic and explain the real-time conditions that led to why the final diagnosis became true. An explanation is created when the final diagnosis becomes true regardless of whether the final diagnosis is high enough to be acted upon. The explanation should not be confused with a text recommendation. The explanation is *why* the final diagnosis is true and the text recommendation is *how* to correct the problem. Another important distinction is that the explanation is not updated if the final diagnosis remains true for an extended period of time whereas quantitative and text recommendations are periodically refreshed.

The capabilities are best demonstrated using the following example diagram:

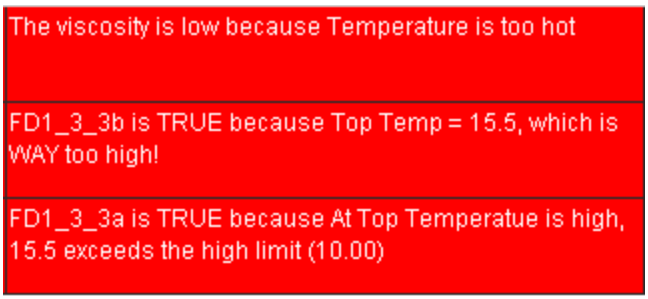


All three final diagnosis become true at the same time. The Diagnosis queue shows the explanation in the Diagnosis column:

Test Console Diagnosis Message Queue

Status	Id	Time	Grade	Application	Family	FP	DP	Diagnosis	Rec Status
Active	5428	2020-09-17 13:03:04	None	TESTAPP1	TestFamily1_3	0	0	The viscosity is low because Temperature is too hot	Made
Active	5429	2020-09-17 13:03:04	None	TESTAPP1	TestFamily1_3	0	0	FD1_3_3b is TRUE because Top Temp = 15.5, which is WAY too high!	Made
Active	5427	2020-09-17 13:03:04	None	TESTAPP1	TestFamily1_3	0	0	FD1_3_3a is TRUE because At Top Temperatue is high, 15.5 exceeds the high limit (10.00)	Made

The same explanations are enlarged below:



The top final diagnosis, FD1_3_3a uses the default explanation facility with is totally automatic using a technique referred to as explanation mining. The algorithm follows truth-value connections, navigating intelligently through Boolean blocks to explain why the final diagnosis is True. The corresponding explanation is the third entry in the diagnosis queue. The explanation includes block names, values, and limits where appropriate. It is worth pointing out that the bottom high limit block, which is False, did not contribute to the final diagnosis being True so it is not mentioned in the explanation.

The bottom two final diagnosis both have an explanation block in the upstream connection path. The explanation block is a pass-through block that is only there to direct the explanation mining. Explanation mining stops searching a branch when it encounters an explanation block. The middle final diagnosis, FD1_3_3b, has an explanation block with a dynamic expression in the explanation. The evaluated expression is the middle entry in the diagnosis queue. The bottom final diagnosis, FD1_3_3c, has an explanation block with a static expression in the explanation. The explanation is the top entry in the diagnosis queue.

The dynamic expression follows the same guidelines that are used for expressions in the sequential control toolkit. The expression is enclosed in "{ }". The only tag that is supported in this environment is tag followed by the tagpath without the tag provider which will be supplied at run time based on the production/isolation state of the diagram.

The explanation mining facility appears to be quite useful in this trivial example. In practice, for a moderately complicated diagram, the explanation mining facility is so detailed that it ends up being difficult to read and comprehend. Therefore the explanation block can be used to make the explanations more readable. It is worth pointing out that multiple explanation blocks can be used, perhaps one on each path leading into an AND block.

4 Symbolic Ai Blocks

This section describes how the blocks work in general and how some blocks that may not be obvious operate.

4.1 Palette Organization

An action step executes a Python script whenever its input value matches the blocks trigger value.

4.2 Block Behavior

With few exceptions, all blocks respond to the following action commands:

propagate – force propagation of the latest values on all outputs

force – place a value on the selected output. Once propagated, the block is placed into a locked state.

lock/unlock – a locked block will not propagate a value on its output(s)

reset – initiate a reset action on the block. Reset behavior is dependent on the block type.

These actions can be triggered via a user menu associated with the block or via a signal connected directly to the block. The signal stub is enabled/disabled interactively by the operator. By default it is not displayed.

4.3 Block Definitions

This section describes specific characteristics of each block type.

Block	Palette	Description
Action	Misc	Execute a configurable Python script
And	Logic	Propagate the logical “and” of the inputs
Arithmetic	Arithmetic	Perform one of many arithmetic functions on the input
Average	Statistics	Maintain an average of all points that have arrived at the block
Average (n)	Statistics	Maintain a rolling average of the most recent n points that have arrived at the block
Average (t)	Statistics	Maintain a time-weighted rolling average of points that have arrived within a time interval
Bias	Arithmetic	Add a specified constant to the input
Change Sign	Arithmetic	Change the sign of the input
Command	Control	Send a specified signal

Block	Palette	Description
Compare	Observation	Compare two data values and return a truth-value
Compare Absolute	Observation	Compare the absolute values of two data inputs and return a truth-value
Compare Deadband	Observation	Compare the values of two data inputs considering a deadband zone.
Conditioner	Connectivity	Apply additional quality constraints to the input
Data Selector	Control	Select one of two inputs depending on a truth-value
Data Shift	Control	Buffer incoming data for a fixed sample size
Data Switch	Control	Send output to one of three ports depending on the value of a control line.
Delay	Timers & Counters	Introduce a delay into the data flow
Difference	Arithmetic	Subtract one input from the other
Discrete Rate Of Change	Arithmetic	Compute the instantaneous rate of change using a least squares fit
Dual Option	Observation	Unimplemented
Edge Trigger	Control	Detect a truth-value change and hold for a specified interval
Equality	Observation	Compare the input to a fixed target value
Explanation	Conclusion	Provide a user-defined explanation for chart state.
Exponential	Arithmetic	Compute ”e” to the value of the input.
Filter	Statistics	Perform an exponentially weighted moving average on the input.

Block	Palette	Description
Final Diagnosis	Conclusion	A specialty block for determining a recommendation based on the sense of the input
Gain	Arithmetic	Multiply the input by a specified constant
High Limit	Statistics	Pass the maximum value observed across a number of inputs, subject to a user-defined maximum
High Limit	Observation	Compare incoming data with a configured upper limit and optional deadband
High Limit (n)	Observation	Return true if m of the last n samples are greater than a specified limit
High Limit (t)	Observation	Return true if n samples within a specified time window are greater than a specified limit
High Selector	Statistics	Pass the maximum value observed across a number of inputs
High Value Pattern	Observation	Return TRUE if ”n” values are above a threshold.
Inference Memory	Control	Remember if an input has ever been true
Inhibitor	Control	Discard input older that a specified date
In Range	Observation	Compare incoming data with a configured range and optional deadband
In Range (n)	Observation	Return true if m of the last n samples were outside a specified range
In Range (t)	Observation	Return true if n samples within a specified time window are within than a specified range
Input	Connectivity	Subscribe to a tag and propagate its value changes
Inverse	Arithmetic	Compute the inverse of the input (1/x).

Block	Palette	Description
Junction	Connectivity	A block that passes information through without change, a null operation
Lab Data	Connectivity	A special input block that reads value and timestamp from separate tags
Linear Fit	Arithmetic	Compute the best fit line over a recent history of data points
Ln	Arithmetic	Compute the natural logarithm of the input
Log10	Arithmetic	Compute the logarithm base 10 of the input
Logic Filter	Logic	Monitor the ratio of time true to time false over a specified interval.
Logic Latch	Logic	Hold a the most recent true/false through a reset
Low Limit	Statistics	Pass the minimum value observed across a number of inputs, subject to a user-defined minimum
Low Limit	Observation	Compare incoming data with a configured lower limit and optional deadband
Low Limit (n)	Observation	Return true if m of the last n samples are less than a specified limit
Low Limit (t)	Observation	Return true if n samples withn a specified time window are less than a specified limit
Low Value Pattern	Observation	Return TRUE if “n” values are below a threshold.
Low Selector	Statistics	Determine the minimum value among inputs.
Median	Arithmetic	Return the median of all inputs
N True	Logic	Return true if a the number of true inputs meets a threshold
Not	Logic	Invert a truth-value

Block	Palette	Description
Note	Misc	A text box for commentary on the diagram
Or	Logic	Propagate the logical “or” of the inputs
Out of Range	Observation	Compare incoming data with a configured range and optional deadband
Output	Connectivity	Write value received on input to a tag.
Parameter	Connection	Encapsulate a tag.
Persistence Gate	Timers & Counters	Guarantees that its input values are unchanged for a specified period
Product	Arithmetic	Compute the product of numerical inputs
Qualified Value	Misc	Create a qualified value from separate value, quality, time streams
Quotient	Arithmetic	Divide the value on one input by the value on the other
Random	Statistical	Inject random noise into the input
Range	Observation	Compare incoming data with a configured range and optional deadband
Readout	Misc	Display the current value of a data connection
Receiver	Connectivity	Accept a signal and propagate on an output connection
Reset	Control	Send a reset signal
Set Property	Control	Set a specified property on a connected downstream block.
SQC	Statistics	Perform a SPC calculation. Execute one of the Westinghouse rules.
SQC Diagnosis	Conclusion	A specialty block for determining a recommendation based on the upstream results of an SQC block.

Block	Palette	Description
State Lookup	Control	Return a truth-value based on selection from a list.
Statistics	Statistics	Compute a selected statistical function on the latest values from each of multiple inputs.
Statistics (n)	Statistics	Apply a selected statistical function to the most recent n points that have arrived at the block
Statistics (t)	Statistics	Maintain a time-weighted statistical function to points that have arrived within a time interval
Sub Diagnosis	Conclusion	A specialty block for determining an action based on an input
Sum	Arithmetic	Add values on a variable number of inputs
Time Fork	Misc	Pass the timestamp of the incoming data on the output
Time Readout	Misc	Display the time of the last value passing through the block
Timer	Timers & Counters	On reset, transmit a configured truth for a specified interval
Trend Detection	Statistics	Apply SQC-like rules to detect trends
Transmitter	Connectivity	Transmit a signal to other blocks, either in the same diagram or not
Truth Counter	Timers & Counters	Count the number of times a trigger value is received
Truth Pulse	Connectivity	Transmit a momentary truth value
Unknown	Logic	Return TRUE if the input is UNKNOWN
XY Fit	Arithmetic	Compute the best fit line over a recent history of pairs of data points
Zero Crossing	Observation	Detect a sign change in the input.

Note that in the observation blocks, a TRUE value indicates an abnormal situation. Time intervals are always expressed in seconds as a double value. This allows fractional seconds to be used, usually for testing.

4.4 Block Descriptions

4.4.1 Action - Misc

An action block executes a Python script whenever it receives a new value on its input that matches the trigger value property. The arguments passed to the script are: the block, tag provider, database. The tag provider and database that are supplied are aware of the state (production or isolation) of the diagram. The Application Programmer's Interface for accessing the internals of Symbolic Ai are described in section 9.

Properties:

Script	full package and module name of the script to be executed.
Trigger	truthvalue that, when received, causes the block to execute the script

Connections:

in	incoming truth value.
out	truth-value, pass the input through the block.

A sample Python script is shown below:

```
'''
Demonstration of a custom action module
'''
def act(block, provider, database):
    print "demo.act block class = ",block.getClassName()
    print "Block ID = ",block.blockId
    print "Block Parent Id = ",block.parentId
    print "Block database = ",database
    print "Block Tag Provider =",provider
```

4.4.2 And - Logic

Propagate the logical "and" of the inputs. This block expects multiple connections on its input port. It accommodates any number. The output is given a GOOD quality unless the block state is UNKNOWN. In that case the quality is set to the "worst" quality of any of the inputs. On a reset, existing values are retained, and an UNKNOWN value is assigned the output state. However, no new output is emitted until the next new value is received.

Properties:

SyncInterval	a “coalescing interval”. This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 (no synchronization).

Connections:

in	truthvalues, multiple upstream blocks may be connected to the single input.
out	truthvalue representing the “and” of the inputs. The timestamp is updated.

4.4.3 Arithmetic - Arithmetic

Apply a custom arithmetic function. The function is written in Python and must accept a qualified value for its input. It must generate a qualified value for the output. The custom code has full access to the Apache commons-math library. There is no reset behavior. The function is designed to be user-modifiable. The current version is found in external python, in *ils.block.arithmetic.py*. The code sample shown in Appendix D is an initial implementation encompassing the following functions:

- abs
- round
- floor
- ceiling
- sine
- cosine
- tangent

Properties:

Function	name of the function to be executed. This is a string value chosen from the list of available functions.

Connections:

in	incoming data value.
out	double value, the function result. The output timestamp is updated.

4.4.4 Average – Statistics

There are three blocks that calculate the average.

4.4.4.1 Average - Statistics

Compute the average of all values that have arrived on the block's input over its entire existence. Report the average on the arrival of each new point. Values of bad quality are ignored for the purpose of the average. When bad value is received, then the output is given a bad quality and the buffer is cleared. The average may, optionally, be cleared on reset.

Properties:

ClearOnReset	If TRUE, the average computation will re-start when the block is reset. By default this property is FALSE.

Connections:

in	incoming data value.
out	data, the accumulated average.

4.4.4.2 Average (n) - Statistics

Compute a rolling average of a specified number of the most recent values that have arrived on the block's input. Once the buffer is filled, report the average on the arrival of each new point. Values of bad quality are ignored for the purpose of the average. When bad value is received, then the output is given a bad quality. The average may, optionally, be cleared on reset.

There is no individual weighting applied to samples. This means that the block is most appropriate where the input is constantly changing and sampled at regular intervals.

Properties:

ClearOnReset	If TRUE, the average computation will re-start when the block is reset. By default this property is FALSE.
SampleSize	the number of points to include in the average. When the sample size is changed, the buffer is cleared

Connections:

in	incoming data value.
out	data, the rolling average.

4.4.4.3 Average (t) - Statistics

Compute the average of all values that have arrived on the block's input within a specified time window. Report the average on the sample interval. Values of bad quality are ignored for the purpose of the average. When a bad value is received, then the output is given a bad quality. The average may, optionally, be cleared on reset.

Properties:

ClearOnReset	If TRUE, the average computation will re-start when the block is reset. By default this property is FALSE
ScanInterval	the sampling interval ~ secs
TimeWindow	the period over which the average is to take place. ~ secs

Connections:

in	incoming data value.
out	data, the accumulated average

4.4.5 Bias - Arithmetic

Change the value of the input by a configured constant and propagate the result. Only a single input connection is expected. There is no synchronization issue, nor reset behavior.

Properties:

Bias	the value offset.

Connections:

in	data value.
out	data value that is the input plus a specified factor. The timestamp is not altered from the incoming value.

4.4.6 Change Sign - Arithmetic

Multiply the input by minus one and propagate the result. Only a single input connection is expected. There is no synchronization issue, nor reset behavior.

Connections:

in	data value.
out	data value, input value multiplied by minus one. The timestamp is not altered from the incoming value.

4.4.7 Command - Control

When a triggering condition is detected on the input or the signal connection receives a START, this block propagates a configurable command on the output. This block is a generalization of the Reset block allowing an arbitrary command.

Properties:

Command	the command to be sent. By default the signal is “reset”.
Trigger	the value to be matched on input in order to trigger the output. Default value is TRUE.

Connections:

in	a truth value. This triggers the output if it matches the trigger value.
out	the output signal.

4.4.8 Compare Blocks - Observation

There are three compare blocks that that have common characteristics. The blocks all compare two numeric or date values, “x” and “y”. The output is a truth-value, true if x >= y. On reset the inputs are set to not-a-number. This requires that new values must be received on both inputs before the block will emit a new result.

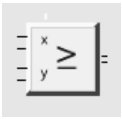
The behavior of the three comparison blocks is summarized below:

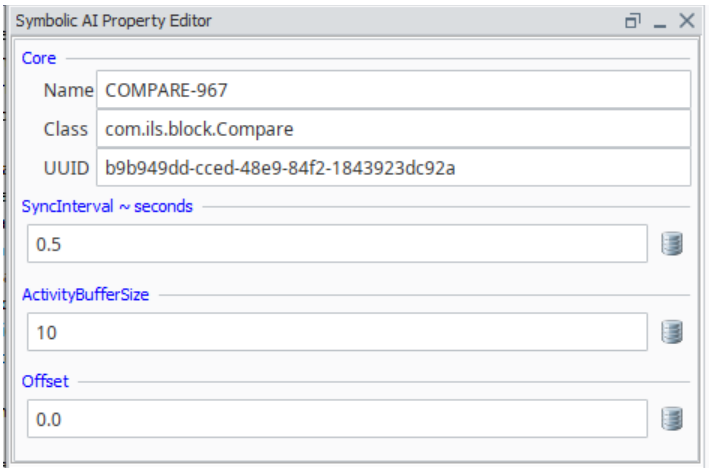
Block	True	False
Compare	X >= Y	X < Y
Compare Absolute	Abs(X) >= abs(Y + deadband)	Abs(X) < abs(Y + deadband)
Compare Deadband	X >= Y	X < Y – abs(deadband)

The inputs may be either numeric values or dates. Comparing a numeric value to a date is not allowed. For a date, “greater than” implies more recent.

4.4.8.1 Compare – Observation

The Compare block, from the “Observation” palette, compares two numeric or date values, “x” and “y”. The output is a truth-value, true if x>=y. On reset the inputs are set to not-a-number. This requires that new values must be received on both inputs before the block will emit a new result.





Properties:

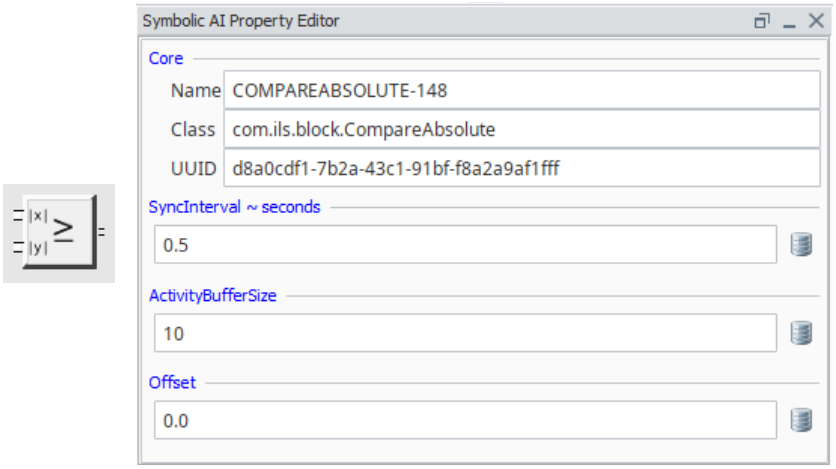
ActivityBufferSize	the number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
Offset	an offset added to "Y" before the comparison. Default value is 0. If dates are being compared, this value is interpreted as milliseconds.
SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 1 sec.

Connections:

x	data connection, the top input.
y	data value, the bottom input.
out	truthvalue connection, the result.

4.4.8.2 Compare Absolute - Observation

The Compare Absolute block, from the "Observation" palette, is identical to the Compare block with the exception that it compares the absolute value of two numeric or date values, "x" and "y". The output is a truth-value, true if abs(x) >= abs(y). On reset the inputs are set to not-a-number. This requires that new values must be received on both inputs before the block will emit a new result.



Properties:

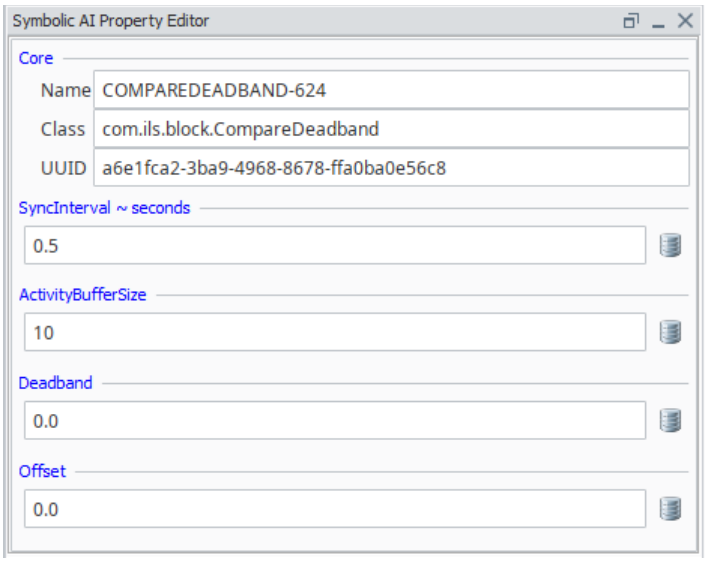
ActivityBufferSize	the number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
Offset	an offset added to "Y" before the comparison. Default value is 0. If dates are being compared, this value is interpreted as milliseconds.
SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 1 sec.

Connections:

x	data connection, the top input.
y	data value, the bottom input.
out	truthvalue connection, the result.

4.4.8.3 Compare Deadband - Observation

The Compare Deadband block, from the "Observation" palette, compare two input data values, "x" and "y". The output is a truth-value, true if x >= y and false if x < y – deadband, where the deadband is a positive configured constant. If the value lands within the deadband zone, then return the previous value. On reset the inputs are set to not-a-number. This requires that new values must be received on both inputs before the block will emit a new result.



Properties:

ActivityBufferSize	the number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
Deadband	a value that defines a tolerance region below "Y". Default value is 0.
Offset	an offset added to "Y" before the comparison. Default value is 0. If dates are being compared, this value is interpreted as milliseconds.
SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 1 sec.

Connections:

x	data connection, the top input.
y	data value, the bottom input.
out	truthvalue connection, the result.

4.4.9 Conditioner - Connectivity

Allow only good quality values to propagate. The quality is determined by the quality of the incoming value and by the state of a secondary quality line. A Boolean value of FALSE (no trouble found) implies a good quality. A result is propagated only if the main value is of good quality and the secondary input is itself of good quality and not TRUE. On initialization and reset, the secondary input is internally set to UNKNOWN. The timestamp of the output is the same as the incoming data value.

Properties:

SyncInterval	a “coalescing interval”. This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 sec.

Connections:

V	incoming truth value.
Q	truth-value, pass the input through the block.
out	data value, a qualified value with always “good” quality
status	truth-value. TRUE if a problem is detected regarding the data quality

4.4.10 Data Selector - Control

Allow one of two inputs to propagate depending on the value of a third input, a truth-value. If the truth-value has a value of TRUE, any incoming value on the first input is propagated. If it has a value of FALSE, then any value on the second input is propagated. Otherwise, no value is propagated. On reset, the control line is set to UNKNOWN.

Properties:

Script	full package and module name of the script to be executed.
Trigger	truthvalue that, when received, causes the block to execute the script

Connections:

in1	data value, the top data line which will be propagated when the control value is True.
in2	data value, the bottom data line which will be propagated when the control value is False.
control	truth-value, determines which input is propagated.
out	data value, the value on in1 or in2 based on the control value.

4.4.11 Data Shift - Control

Store and hold incoming data values in a fixed size FIFO buffer. When the buffer fills, additional incoming data are placed at the buffer head and the oldest data are placed on the output connection. A reset action clears the buffer. Values with bad quality are ignored.

Properties:

SampleSize	The number of input values to store, the default value is 0 – which results in pass-through behavior.

Connections:

in	data to be processed.
out	data released when the buffer is full. The original timestamps of the data values are retained.

4.4.12 Data Switch - Control

Route the input to one of three ports (true, unknown, false) depending on the value of a control line. Change in the control value does not cause a value to propagate, however a synchronization interval is provided to prevent the race condition when data value and control value are sent nearly simultaneously. If the control value is UNSET, then no value is propagated. To operate in a True/False only mode, simply leave the "unknown" port unconnected.

Properties:

SyncInterval	A "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 (no synchronization).

Connections:

in	Input value, any datatype is allowed.
control	truth value, determines which output is active
out (top)	output when the control is TRUE.
out (middle)	output when the control is UNKNOWN.
out (bottom)	output when the control is FALSE.

4.4.13 Delay – Times & Counters

Store and hold incoming data values for a specified period. The block allocates memory as necessary in order to hold values waiting for their respective timeout. This block accepts any input type. A reset action deletes all queued information.

Properties:

SampleDelay	the length of time for which input values are delayed. ~ secs. The default value is 0.

Connections:

in	raw data to be analyzed. Any datatype is accepted.
out	data, truthvalue or string. The output is the input value without change (just delayed). The original timestamps of the data values are retained.

4.4.14 Difference - Arithmetic

Compute the difference of two inputs. Report the value a port "a" minus the value of the port labeled "b". On a reset, all existing values are forgotten. If the data types of both inputs are Date, then the result will be the difference in time between the inputs, expressed as seconds. If only one of the input data types is a date, the output will be NaN with a bad quality.

Properties:

Script	full package and module name of the script to be executed.
Trigger	truthvalue that, when received, causes the block to execute the script

Connections:

a	data value. The number to subtract from
b	data value. The number to subtract
out	double value representing the difference of the inputs. The output timestamp is updated

4.4.15 Discrete Rate of Change - Arithmetic

Using a least squares fit (linear or quadratic) on a specified number of the most recent values that have arrived on the block's input, report the slope for the most recent arrival on the "fit" port. The scaled slope is reported on the "slope" port.

Properties:

ClearOnReset	If TRUE, the linear fit calculations will re-start when the block is reset. By default this property is FALSE.
NumberOfPoints	5 or 7. These are the only legal values
PolynomialOrder	2 or 3. These are the only legal values
ScaleFactor	report the best fit slope multiplied by this number. Default value is: 1.0

Connections:

in	data connection, the raw input.
out	instantaneous slope at the most recent point, multiplied by the scale factor.

4.4.16 Edge Trigger - Control

When an incoming truth-value matches the trigger value, then the output reflects that truth-value for the specified interval before returning to the inverse value. There is no synchronization issue. On reset the timer is cancelled and the prior value is propagated to the output. Initially the value is unset until a new value arrives on the input (and then that value becomes the output). New values arriving during the hold period are ignored. Changing the hold interval does not affect any interval in progress.

Properties:

HoldInterval	The interval ~secs during which the output is delayed. Default is zero, which results in a short pulse
Trigger	the incoming value which starts the hold interval

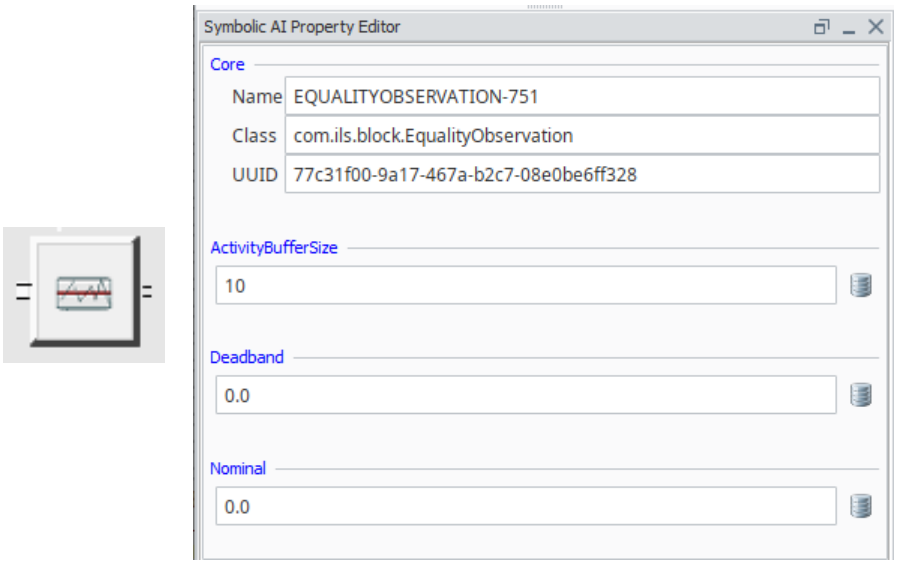
Connections:

in	truth value.
out	input value held for the specified interval, then returned to its inverse

4.4.17 Equality – Observation

The Equality block, from the "Observation" palette, compares the input against a configured target value and within a specified deadband as shown by the truth-table below. There are no synchronization considerations, nor reset behavior.

TRUE	FALSE
X >= Nominal – Deadband/2 and X <= Nominal + Deadband/2	X < Nominal – Deadband/2 or X > Nominal + Deadband/2



Properties:

ActivityBufferSize	the number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
Deadband	a tolerance region around the target. Default value is 0
Nominal	the nominal target value. The default is zero.

Connections:

in	data connection, the input
out	truthvalue connection, the result.

4.4.18 Explanation - Conclusion

Provide a facility for a user to enter a custom explanation for the truth-value of downstream blocks. The automatically generated explanation may be too complex to be understandable, depending on the diagram. Other than for explanations, the block behaves as a pass-thru. There is no synchronization, nor reset behavior.

Properties:

explanationWhenFalse	text, the explanation to be used when the block is FALSE
explanationWhenTrue	text, the explanation to be used when the block is TRUE.

Connections:

in	incoming truth value.
out	truth value, the input value

4.4.19 Exponential - Arithmetic

Raise “e” to the value of the input and propagate the result. Only a single input connection is expected. There is no synchronization issue, nor reset behavior.

Connections:

in	data value
out	data value that is e to the power of the input. The timestamp is not altered from the incoming value.

4.4.20 Filter - Statistics

Smooth the input with an exponentially-weighted-moving-average. There is one output for each input. On reset, the average calculation is initialized. Bad data is ignored in the moving average. Entries are weighed by the time-interval between inputs.

Properties:

TimeWindow	a time constant for the average ~ secs

Connections:

in	incoming data value.
out	double value representing the input filtered as a moving average. The output timestamp is updated.

4.4.21 Final Diagnosis - Conclusion

Refer to section *2.3 Problem / Final Diagnosis* for a conceptual discussion of this block. This block has a single Boolean input. When it becomes *True* it signifies that a problem has been detected and the process for rectifying the problem is initiated. If the problem is the highest priority active problem for the family then the recommendation configured for the final diagnosis is implemented.

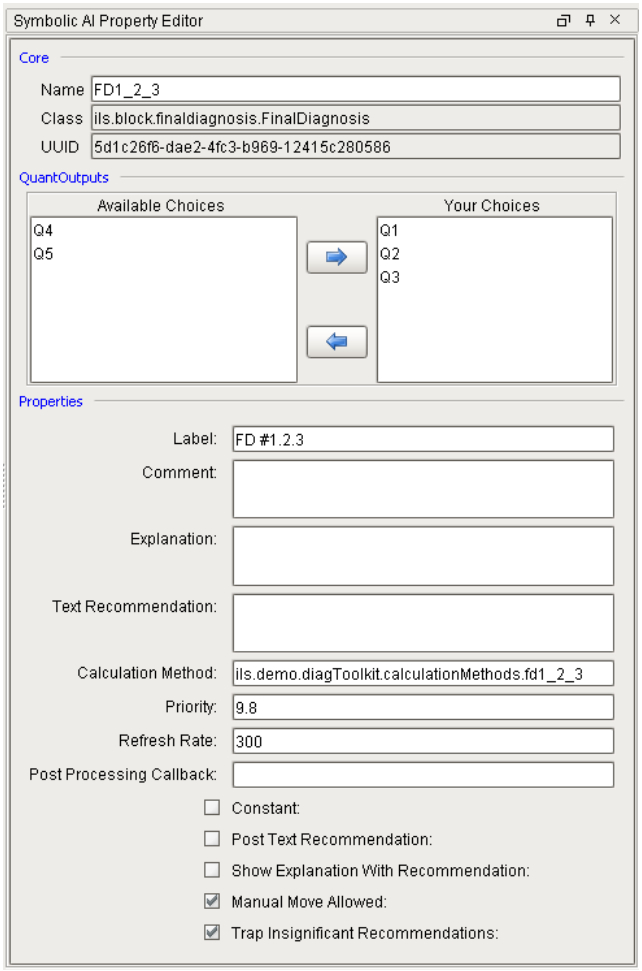
The recommendation may take three forms:

1. Quantitative: specific numeric recommendations to process control variables. The numeric recommendations are determined dynamically as specified in the calculation method which is called by the management framework when the Final Diagnosis becomes true.

2. Text: instructions of manual actions to take to correct the problem. Text recommendations may be statically configured in the *Text Recommendation* property or they may be dynamically configured in the calculation method.
3. Constant: a constant final diagnosis executes silently, it does not make a numeric or text recommendation, rather, it locks out all lower priority problem, typically used for unit shutdown or changeover situations.

Calculation methods are discussed in detail in section *3.6 Calculation Method*

The property editor for a final diagnosis is a bit more complicated than a normal property editor and is shown below:



Properties:

Calculation method	The name of a Python, generally in external Python, for making quantitative and text recommendations.
Comment	Freeform comment purely for documentation purposes

Constant	Indicates a state of the plant that cannot be changed. The Final Diagnosis will remain active until the conditions that lead up to it change. A good example is idle, shutdown, or not running. Constant final diagnosis are typically given a very high priority which would preclude running any other diagnostic diagrams. A constant final diagnosis does not make text or numeric recommendations.
Explanation	Used for information in the diagnosis field of the diagnosis entry that is added to the Diagnosis Queue whenever the final diagnosis becomes True.
Manual Move Allowed	True or False. If True, then this final diagnosis will appear in a list of manually triggering a final diagnosis. This allows the full calculation, notification and download frameworks to be used for an operator initiated event.
Priority	The priority of the final diagnosis. The diagnosis manager will ensure that only the highest priority final diagnosis will be acted upon in the case of simultaneous problems.
Post Processing Callback	Specifies the name of a Python script that will be called after a text recommendation has been acknowledged or numeric recommendations have been downloaded. This runs in the client and can be used to display a window or perform any other action that may be relevant to the problem being solved.
Post Text Recommendation	True or False, if True then a text recommendation will be posted, otherwise a quantitative recommendation is expected.
Quant Outputs	The set of quant outputs for which a recommendation may be made. The set is a subset of the quant outputs defined for the application. The recommendations made at a particular time do not need to include every quant output in the set, but they cannot include a quant output that is NOT defined in the set.
Refresh rate	The rate at which the recommendations are refreshed by calling the calculation method. Default is 5 minutes.
Show explanation With Recommendation	True or False, if True and the Post Text Recommendation box is checked and there is an explanation, then the explanation and the text recommendation will be concatenated and displayed to the operator.
Text Recommendation	A static text recommendation that is posted to the operator if *Post Text Recommendation* is True
Trap Insignificant recommendations	True or False, if True then insignificant recommendations, as determined by the configuration of the quant output, will be ignored.

Connections:

in	incoming truth value.
out	truth-value, pass the input through the block.

4.4.22 Gain - Arithmetic

Multiply the value of the input by a configured constant and propagate the result. Only a single input connection is expected. There is no synchronization issue, nor reset behavior.

Properties:

Gain	the multiplier

Connections:

in	incoming data value.
out	data value that is a specified factor times the input. The timestamp is not altered from the incoming value.

4.4.23 High Limit - Statistics

Consider values on a variable number of inputs. Pass the maximum value subject to a specified upper limit. The output is given a GOOD quality if the value on any input is GOOD. Values of bad quality are ignored. On a reset, existing values are retained. However, no new output is emitted until the next new value is received. If a new value is not received within the scan interval, then a value for that input is generated as a copy of the last good value.

Properties:

SyncInterval	a “coalescing interval”. This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 (no synchronization).

Connections:

in	data values. Multiple upstream blocks may be connected to the single input.
out	data representing the maximum of the inputs. The timestamp is updated.

4.4.23 High Limit Blocks - Observation

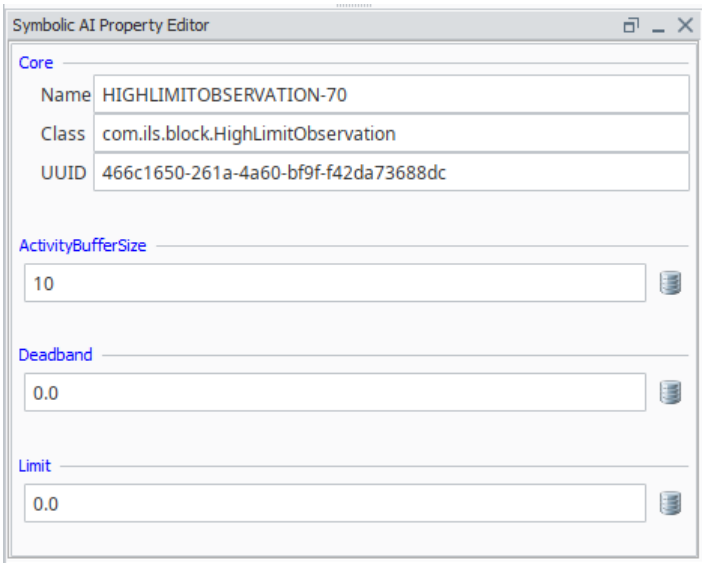
There are three High limit observation blocks.

4.4.23.1 High Limit – Observation

The High Limit Observation block, from the “Observation” palette, compares incoming input data values against a configured upper limit. The output is a truth-value determined

per the table below. For values that fall within the deadband range, retain the most recent past value. On reset the past value is set to UNKNOWN.

TRUE	FALSE
X >= Limit	X < Limit - Deadband



Properties:

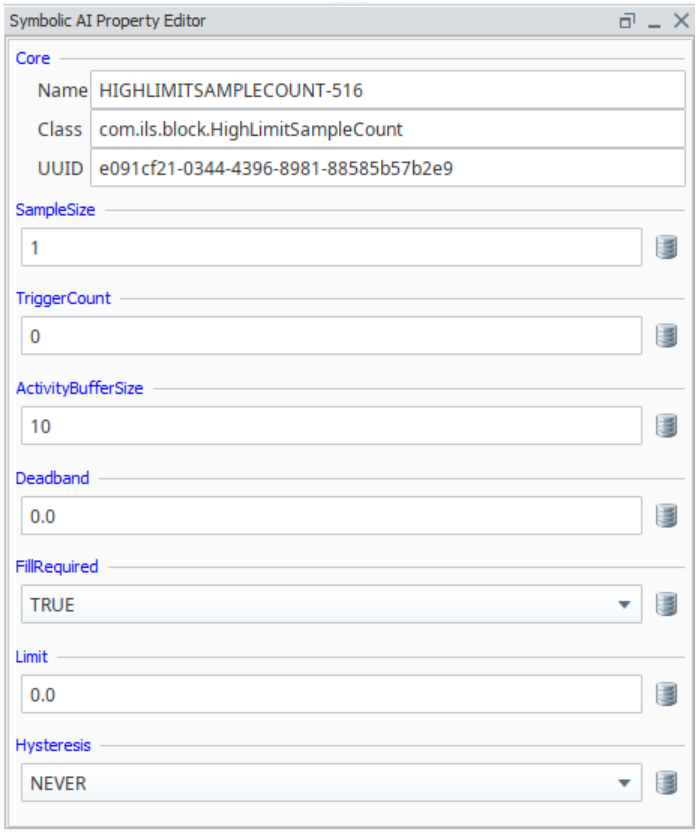
ActivityBufferSize	the number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
Deadband	the value of the deadband, a non-negative number. Default = 0
Limit	the value of the limit. Default = 0.

Connections:

in	data, the raw input
out	truthvalue, the result.

4.4.23.2 High Limit (Sample Count) – Observation

The High Limit Sample Count block, from the "Observation" palette, compares the last "Sample Size" values to specified limit. If the number of values greater than the limit is greater than the trigger limit then the output of the block is TRUE. For the period of time that the buffer is not filled and a FALSE cannot be assured, the result is dependent on the FillRequired property. If TRUE then the conclusion is UNKNOWN; otherwise the conclusion is FALSE.



Properties:

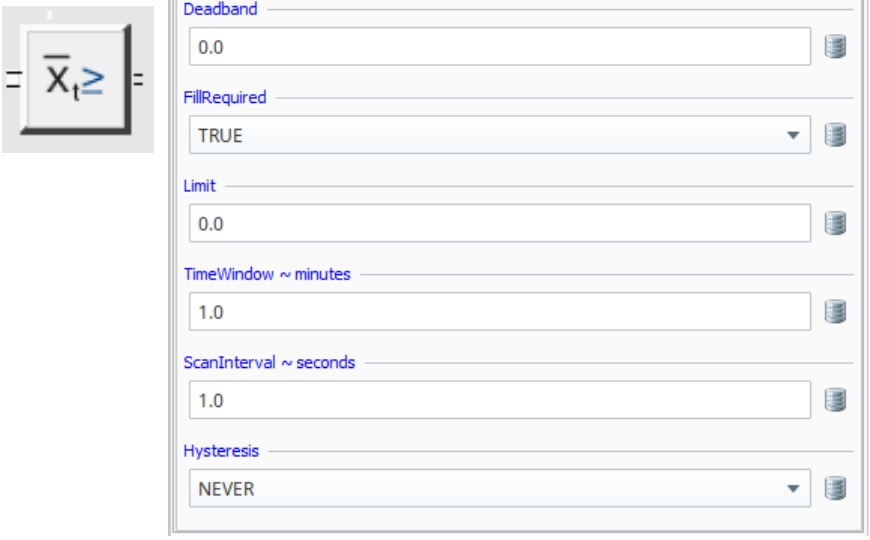
ActivityBufferSize	The number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
Deadband	The value of the deadband, a non-negative count. If the value is zero, then there is no deadband consideration. Default = 0
Hysteresis	TRUE, FALSE, ALWAYS, NEVER. Default is NEVER. Refer to the "LogicFilter" block for details.
Limit	The value of the limit, default is zero
FillRequired	If TRUE, the state of the block is UNKNOWN until the buffer fills. Default is TRUE.
TriggerCount	The minimum number of samples that must be greater than the limit in order to evaluate as TRUE. Default = 0
SampleSize	The number of points to include in the analysis. When the sample size is changed, the buffer is cleared

Connections:

in	data, the raw input
out	true if the input is true for more than the limit

4.4.23.3 High Limit (t) – Observation

The High Limit Time Window block, from the "Observation" palette, compares input values within a specified time window for values greater than a specified limit. If a new value is not received within the scan interval, then a value for that interval is generated as a copy of the last good value. Values of bad quality are ignored.



Properties:

ActivityBufferSize	The number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
Deadband	The value of the deadband, a non-negative count. If the value is zero, then there is no deadband consideration. Default = 0.
FillRequired	If TRUE, the state of the block is UNKNOWN until the buffer fills. Default is TRUE.
Hysteresis	TRUE, FALSE, ALWAYS, NEVER. Default is NEVER. Refer to the "LogicFilter" block for details.
Limit	The value of the limit, default is zero
ScanInterval	The sampling interval ~ secs
TimeWindow	The period over which the analysis is to take place. ~ secs
TriggerCount	The minimum number of samples that must be less than the limit in order to evaluate as TRUE. Default = 0.

Connections:

in	data, the raw input
out	true if the input is true for more than the limit

4.4.24 High Selector - Statistics

Pass the maximum value observed across multiple inputs and propagate the result. Only values with good quality are considered. Values of bad quality are ignored. The timestamp of the output is taken from the maximum input. If multiple inputs have the same maximum the timestamp is indeterminate.

How is this different than the High Limit – Statistics block??

Properties:

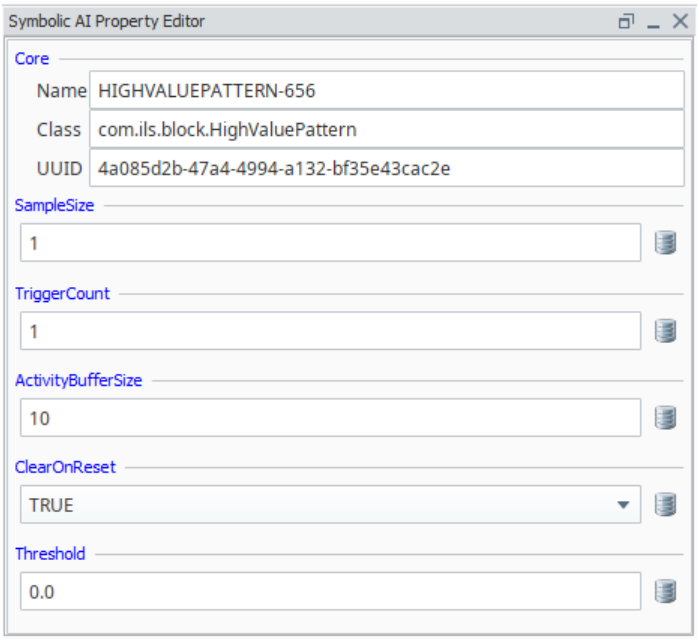
SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0.5 sec

Connections:

In	incoming truth value.
Out	data value that is the maximum of all input values. When a maximum is propagated, the timestamp is equal to the timestamp of the input that is the maximum

4.4.25 High Value Pattern – Observation

The High Value Pattern block, from the "Observation" palette compares the last "n" input values against a specified threshold. Return TRUE if the number of values above the threshold equals or exceeds a trigger count. For the initial time period where the buffer is not filled and a result cannot be determined, the conclusion is UNKNOWN.



Properties:

ActivityBufferSize	The number of activities to record in the blocks buffer which is useful for debugging. Default is 10.
ClearOnReset	If TRUE, the calculations will re-start when the block is reset. By default this property is TRUE.
Threshold	The value of the limit, default is zero
SampleSize	The number of points to include in the analysis. When the sample size is changed, the buffer is cleared
TriggerCount	The minimum number of samples that must be greater than the threshold in order to evaluate as TRUE. Default = 1.

Connections:

In	data, the input being monitored
Out	truth value, true if trigger count inputs exceed the threshold

4.4.26 In Range - Observation

There are several in range observation blocks.

4.4.26.1 In Range (n) - Observation

Test the last "n" samples input for values within a specified range. Return TRUE if the number of values outside the range exceeds a trigger count. For the period of time that the buffer is not filled and a FALSE cannot be assured, the result is dependent on the FillRequired property. If TRUE then the conclusion is UNKNOWN; otherwise the conclusion is FALSE.

Properties:

Deadband	the value of the deadband, a non-negative count. If the value is zero, then there is no deadband consideration. Default = 0
Hysteresis	TRUE, FALSE, ALWAYS, NEVER. Default is NEVER Refer to the "LogicFilter" block for details.
LowerLimit	the value of the lower limit, default is zero
FillRequired	if TRUE, the state of the block is UNKNOWN until the buffer fills. Default is TRUE
SampleSize	the number of points to include in the analysis. When the sample size is changed, the buffer is cleared
TriggerCount	the minumum number of samples that must be in-range in order to evaluate as TRUE. Default = 0
UpperLimit	the value of the upper limit, default is zero

Connections:

In	data, the input being monitored
Out	truth value, true if the input is true for more than the limit.

4.4.26.2 In Range (t) - Observation

Test inputs within a specified time window for values within a specified range. Return TRUE if the number of values outside the range exceeds a trigger count.For the period of time that the buffer is not filled and a FALSE cannot be assured, the result is dependent on the FillRequired property. If TRUE then the conclusion is UNKNOWN; otherwise the conclusion is FALSE. If a new value is not received within the scan interval, then a value for that connection is generated as a copy of the last good value.

Properties:

Deadband	the value of the deadband, a non-negative count. If the value is zero, then there is no deadband consideration. Default = 0
Hysteresis	TRUE, FALSE, ALWAYS, NEVER. Default is NEVER. Controls the behavior with regard to deadband processing. Refer to the "Logic Filter" block for details.
LowerLimit	the value of the lower limit, default is zero.
ScanInterval	the sampling interval ~ secs
TimeWindow	the period over which the analysis is to take place. ~ secs
TriggerCount	the minumum number of samples that must be less than the limit in order to evaluate as TRUE. Default = 0.
UpperLimit	the value of the upper limit, default is zero

Connections:

in	data, the input being monitored
out	truth value, true if the count is greater than the limit

4.4.27 Inference Memory - Control

The Inference Memory block remembers whether the top input (marked "S") has ever been TRUE since either the block has been started or reset path activated. The top input, marked "S," is the set path, and the bottom input, marked "R," is the reset path. When the set path becomes TRUE and the reset path is already FALSE, the block's output becomes TRUE. When the reset path becomes TRUE, the block's value is "reset" to FALSE.

When the block evaluates its inputs, it goes through the steps in this list in the order shown to determine what value to pass. Inputs that have never received a value are treated as "UNKNOWN".

If R is TRUE, it passes FALSE.

If R is UNKNOWN, it passes UNKNOWN.

If S is TRUE, it passes UNKNOWN.

If previous output is TRUE, it passes TRUE.

If S is UNKNOWN, it passes UNKNOWN.

On startup or on block reset, it passes UNKNOWN.

Connections:

set	truth-value, the set input
reset	truth-value, the reset line
out	truth-value, the result

4.4.27 Inhibitor - Control

An inhibit block is used to stop the flow of data through a diagram. The block can be placed inline in any connection type and works the same on Boolean, integer, and float values. It discards any input that has a time-stamp more recent than a specified target. This block has two main modes of operation. First, it can be controlled by a boolean connection at the top of the block. Second, it responds to an "inhibit" signal that is sent to the block either by the scripting API or by a transmit and receive block connected to the signal path. A typical use of this block is for discarding laboratory measurements that may not apply to the current control regime.

Properties:

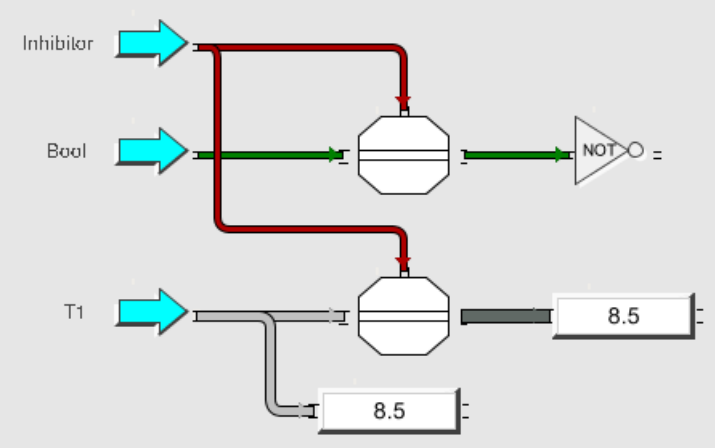
InhibitInterval	time period measured from receipt of a triggering signal during which to discard input ~ secs. The default value is 0. A negative value sets the gating time in the past. This may still be appropriate to discard data arriving with time-stamps in the past. This property has no affect when the block is controlled by a Boolean connection at the top port.

Connections:

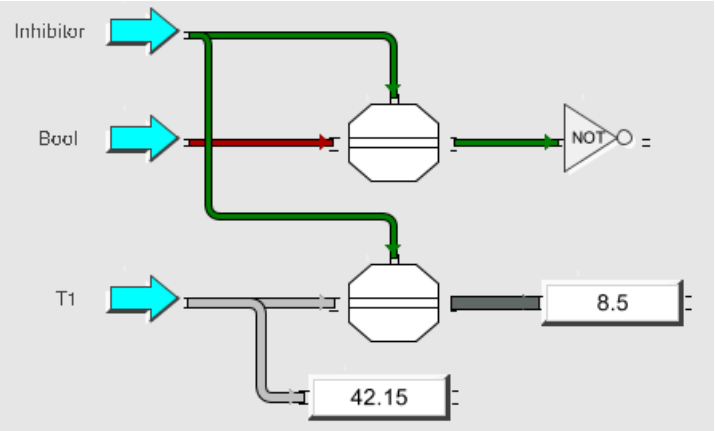
in	Incoming - data connection.
control	Incoming – Boolean connection that provides explicit control over the inhibit behavior.
out	Outgoing - data connection. When the block is operating in pass-thru mode, incoming data are propagated without change to the timestamp

The figure below shows two inhibit blocks, the top one operating on Boolean data and the bottom one on float data. Both blocks are controlled by the Boolean input at the control

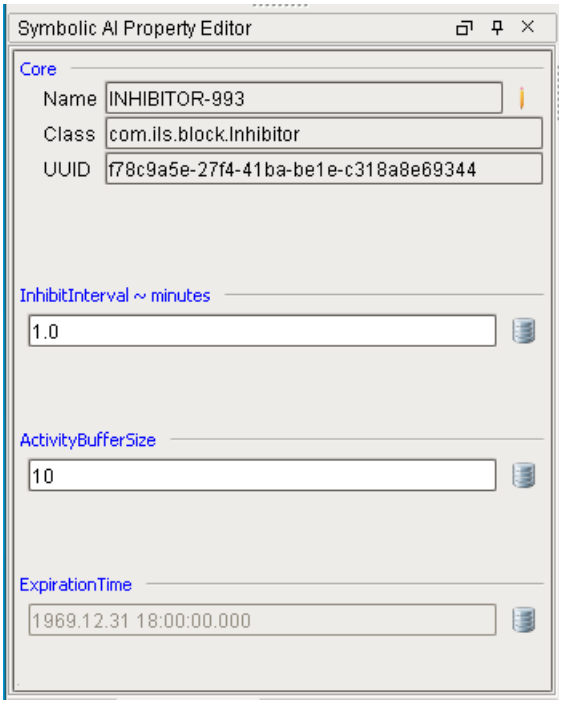
port on top of the block. The inhibitor is False which allows the data to flow through the block as shown by the output of the block being the same as the input of the block.



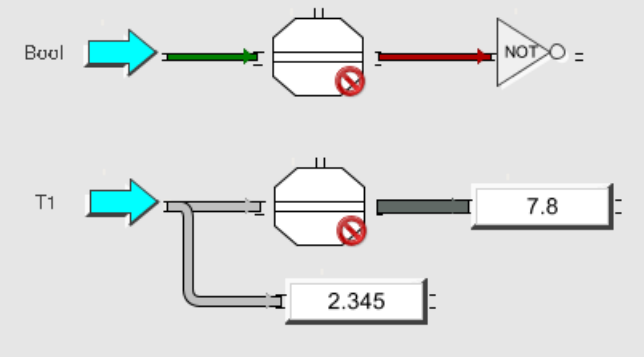
The diagram below shows the blocks while they are inhibiting data flow. The inhibit input is True which means that the block is inhibiting data flow as shown by the output values are different than the input values.



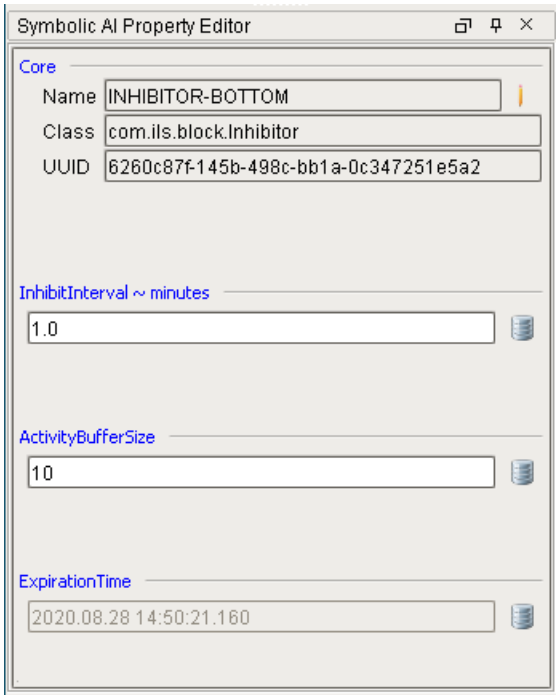
When an Inhibitor block is controlled from top control port then the *InhibitInterval* property does not apply. The *ExpirationTime* also does not apply because the block will stop inhibiting when the control input becomes **False**.



The second mode of operation, via an *inhibit* signal is shown below. When the block receives this signal, then it will inhibit data flow for the time specified by the *InhibitInterval* property. The block is animated by the addition of the red inhibit badge to reflect when the block is inhibiting data due to the signal. The figure shown below is currently inhibiting data flow through both blocks. The red inhibit badge will be removed when the *InhibitInterval* expires.



The *ExpirationInterval* property will be updated with the time calculated by adding the *InhibitInterval* to the time from when the inhibit signal was received.



The inhibit block plays a crucial role in the performance of Symbolic Ai in a real time application. When a problem is detected and a Final Diagnosis becomes active and makes some numeric recommendations, and the recommendations are accepted and downloaded, then the operating regimen of the plant has changed. An inhibit block is often placed downstream from an input block that brings some measurement of product quality. The *InhibitInterval* of the inhibit block should be configured with a time constant that reflects the time that it takes for the process change to reach the measured value. It is also important to note that when used upstream of a Final Diagnosis, the Inhibitor is automatically set when the recommendations are downloaded.

4.4.28 Input - Connectivity

Subscribe to a tag and transmit new values. On a diagram startup, the tag values are propagated. Shortcut feature: If a tag is dragged from the tag browser and dropped on the left side of the diagram workspace it will become an Input block, if dragged onto the right side it will become an output block.

Properties:

TagPath	should be bound to a tag

Connections:

out	data connection, the tag value

4.4.29 Inverse - Arithmetic

Divide one by the input and propagate the result. Only a single input connection is expected. There is no synchronization issue, nor reset behavior. An incoming value of zero is ignored.

Connections:

in	incoming data value.
out	data value that is one divided by the input. The timestamp is not altered from the incoming value.

4.4.30 Junction - Connectivity

A junction block simply takes what appears on its input and transfers to its output. The purpose of this block is to provide an anchor to guide the auto routed path.

Connections:

in	incoming value.
out	the input value is propagated immediately.

4.4.31 Lab Data - Connectivity

This is a block that creates a qualified value from separate inputs, a value and a timestamp. The datatype of the output is the same as that of the value input. The input determines the default quality and timestamp as well. The block will emit an output for every value received on its input.

The timestamp input has a datatype of either a date or a string. If a string, its format is specified by the FORMAT property of the block. This is a string date-time format as specified by the Java SimpleDateFormat class. The default value is: “YYYY/MM/dd hh:mm:ss”. If the format does not include a date (i.e. is a time only), the current date is assumed. If it does not include a time, the current time is assumed. The timestamp is taken as the last value received on the time input. If no input has ever arrived on the time channel, the timestamp of the output is assigned the timestamp of the value input.

A synchronization interval should be specified to guarantee proper pairing of readings between the separate input channels. The synchronization is driven by the “value” input only. This prevents re-issuance of a value on receipt of quality or timestamps only.

Properties:

Format	string format for the date-time. If the "time" input is textual (as opposed to a date) it is converted to a timestamp via this format
SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 1 sec.
TimePath	tag path of the timestamp input. The data type may be either a date-time or string.
ValuePath	tag path of the value input. To make any sense, this value should be bound to a tag.

Connections:

out	qualified value constructed from the value and time paths.

4.4.32 Linear Fit - Arithmetic

Compute the least squares fit from a specified number of the most recent values that have arrived on the block's input. Once sufficient values (2) have been received, report the best fit value for the most recent arrival on the "fit" port using the least squares calculation. The scaled slope is reported on the "slope" port.

Properties:

ClearOnReset	If TRUE, the linear fit calculations will re-start when the block is reset. By default this property is FALSE
FillRequired	If TRUE, the linear fit calculation does not take place until the number of points determined by the sample size. Otherwise only 2 points are required
SampleSize	the number of points to include in the calculation. When the sample size is changed, the buffer is cleared.
ScaleFactor	report the best fit slope multiplied by this number. Default value is: 1.0.

Connections:

in	data, the raw input
out	best fit value for the current input
slope	slope over the current history, multiplied by the scale factor

4.4.33 Ln - Arithmetic

Take the natural logarithm of the input and propagate the result. There is no synchronization issue, nor reset behavior. Input values less than or equal to zero are ignored.

Connections:

in	incoming data value.
out	data value that is the natural logrithm of the input. The timestamp is not altered from the incoming value.

4.4.34 Log10 - Arithmetic

Take the logarithm base 10 of the input and propagate the result. There is no synchronization issue, nor reset behavior. Input values less than or equal to zero are ignored.

Connections:

in	incoming data value.
out	data value that is the logarithm base 10 of the input. The timestamp is not altered from the incoming value.

4.4.35 Logic Filter - Logic

Test the input for amount of time TRUE versus amount of time FALSE. Report TRUE if this ratio is greater than a specified ratio within a specified time interval. If sufficient time to conclude a TRUE result has not yet passed since the last reset, the block state is UNKNOWN.

The “hysteresis” property controls the behavior with regards to the deadband as follows:

```
ratio = sum of time true / timeWindow

case (hysteresis) of
      true:
            if currentOutput = True then
                  threshold = limit – deadband
            else
                  threshold = limit

      false:
            if currentOutput = True then
               threshold = limit
            else
               threshold = limit + deadband

      always:
            if currentOutput = True then
               threshold = limit - deadband
            else
               threshold = limit + deadband

      never:
            threshold = limit

end;
```

```
if ratio > threshold then
   output = True
else
   output = False
```

Properties:

Deadband	the value of the deadband, a fractional, non-negative number. If the value is zero, then there is no deadband consideration. Default = 0
Hysteresis	TRUE, FALSE, ALWAYS, NEVER. Default is NEVER
Limit	the minumum ratio of time that the input is true versus false Default = 0.
Ratio	the current fraction of time-true to the total time. This property has an ENGINE binding and is, therefore, visible in the designer by subscription
ScanInterval	the sampling interval ~ secs
TimeWindow	a sliding time window over which the ratio is calculated ~ secs

Connections:

in	truth value, the input being monitored
out	truth value, true if the input is true for more than the limit

4.4.36 Logic Latch - Logic

The function of the latch block is to preserve TRUE or FALSE values following a diagram reset. The block does not respond to UNKNOWN or UNSET. It will propagate only TRUE or FALSE values that are of good quality. Upon reset, the block will re-transmit its latest TRUE or FALSE value. If it has never received a value, as for example after a Gateway restart, it will transmit an UNKNOWN.

Connections:

in	incoming truth value.
out	the latest TRUE or FALSE value

4.4.37 Low Limit - Statistics

Consider values on a variable number of inputs. Pass the minimum value subject to a specified lower limit. The output is given a GOOD quality if the value on any input is GOOD. Values of bad quality are ignored. On a reset, existing values are retained. However, no new output is emitted until the next new value is received. If a new value is not received within the scan interval, then a value for that input is generated as a copy of the last good value.

Properties:

SyncInterval	a “coalescing interval”. This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 (no synchronization).

Connections:

in	data values. Multiple upstream blocks may be connected to the single input.
out	data representing the minimum of the inputs. The timestamp is updated

4.4.38 Low Limit Blocks – Observation

There are three low limit observation blocks.

4.4.38.1 Low Limit - Observation

Validate incoming input data values against a lower limit. The output is a truth-value determined per the table below. For values that fall within the deadband range, retain the most recent past value. On reset the past value is set to UNKNOWN.

TRUE	FALSE
X < Limit	X > Limit + Deadband

Properties:

Deadband	the value of the deadband, a non-negative number. Default = 0.
Limit	the value of the limit. Default = 0.

Connections:

in	data connection, the raw input
out	Truth value, the result

4.4.38.2 Low Limit (n) - Observation

Test the last “n” samples input for values less than a specified limit. For the period of time that the buffer is not filled and a FALSE cannot be assured, the result is dependent on the FillRequired property. If TRUE then the conclusion is UNKNOWN; otherwise the conclusion is FALSE.

Properties:

Deadband	the value of the deadband, a non-negative count. If the value is zero, then there is no deadband consideration. Default = 0.
Hysteresis	TRUE, FALSE, ALWAYS, NEVER. Default is NEVER. Controls the behavior with regard to deadband processing. Refer to the "Logic Filter" block for details
Limit	the value of the limit.
FillRequired	if TRUE, the state of the block is UNKNOWN until the buffer fills. Default is TRUE.
SampleSize	the number of points to include in the analysis. When the sample size is changed, the buffer is cleared. Default is 1.
TriggerCount	the minumum number of samples that must be less than the limit in order to evaluate as TRUE. Default = 0.

Connections:

in	truth value, the input being monitored.
out	truth value, true if the input is true for more than the limit.

4.4.38.3 Low Limit (t) = Observation

Test inputs within a specified time window for values less than a specified limit. For the period of time that the buffer is not filled and a FALSE cannot be assured, the result is dependent on the FillRequired property. If TRUE then the conclusion is UNKNOWN; otherwise the conclusion is FALSE. If a new value is not received within the scan interval, then a value for that interval is generated as a copy of the last good value. Values of bad quality are ignored.

Properties:

Deadband	the value of the deadband, a non-negative count. If the value is zero, then there is no deadband consideration. Default = 0.
Hysteresis	TRUE, FALSE, ALWAYS, NEVER. Default is NEVER. Controls the behavior with regard to deadband processing. Refer to the "Logic Filter" block for details
Limit	the value of the limit, default is zero.
ScanInterval	the sampling interval ~ secs
TimeWindow	the period over which the analysis is to take place. ~ secs.
TriggerCount	the minimum number of samples that must be less than the limit in order to evaluate as TRUE. Default = 0.

Connections:

in	truth value, the input being monitored.
out	truth value, true if the count is greater than the limit.

4.4.39 Low Selector - Statistics

Pass the minimum value observed across multiple inputs and propagate the result. Only values with good quality are considered. Values of bad quality are ignored. The SyncInterval property provides for synchronization across multiple entry connections. Data arriving when the block is locked are ignored. The timestamp of the output is taken from the maximum input. If multiple inputs have the same minimum the timestamp is indeterminate.

Properties:

SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0.5 sec.

Connections:

in	incoming data value.
out	data value that is the maximum of all input values. When a minimum is propagated, the timestamp is equal to the timestamp of the input that is the minimum

4.4.40 Low Value Pattern - Observation

Test the last "n" samples input for values below a specified threshold. Return TRUE if the number of values below the threshold equals or exceeds a trigger count. For the initial time period where the buffer is not filled and a result cannot be determined, the conclusion is UNKNOWN.

How is this different from the low limit Observation???

Properties:

ClearOnReset	If TRUE, the calculations will re-start when the block is reset. By default this property is TRUE.
Threshold	the value of the limit, default is zero.
SampleSize	the number of points to include in the analysis. When the sample size is changed, the buffer is cleared
TriggerCount	the minumum number of samples that must be less than the threshold in order to evaluate as TRUE. Default = 1.

Connections:

in	data, the input being monitored.
out	truth value, true if trigger count inputs exceed the threshold.

4.4.41 Median - Arithmetic

Analyze values received on multiple input connections, compute the median and propagate to the output. This block uses the median() function from Apache commons math library. There is no special reset behavior.

Properties:

SyncInterval	a “coalescing interval”. This is the time to allow data to arrive on multiple inputs. It is also the no-data-change wait interval. The sync interval must be less than the scan interval.~ secs. The default value is 1 sec.

Connections:

in	Multiple data
out	data, the median value of the inputs

4.4.42 N True - Logic

Propagate a “true” if the number of input true values meets or exceeds a specified threshold. This block expects multiple connections on its input port. The output is given a GOOD quality unless the block state is UNKNOWN. In that case the quality is set to the “worst” quality of any of the inputs. On a reset, existing values are retained, and an UNKNOWN value is assigned the output state. However, no new output is emitted until the next new value is received.

Properties:

SyncInterval	a “coalescing interval”. This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 (no synchronization).
NTrue	the number of inputs which must carry a “true” value in order to propagate a “true” on the output. The default value is 0 (always succeeds).

Connections:

in	incoming truth value - multiple
out	truth-value, True if N or more inputs are True.

4.4.43 Not - Logic

Propagate the logical inversion of the input. Only a single input connection is expected. There is no synchronization issue, nor reset behavior.

Connections:

in	incoming truth value.
out	truth-value, representing the "not" of the input. The timestamp is not altered from the incoming value.

4.4.44 Note - Misc

A note block is a place to store commentary regarding the diagram. A note block has no behavior, has no connections and does not participate in the execution of the diagram. If the text is legal HTML, then the HTML is used for formatting. Like any other attribute, the text may be linked to a tag for live updating. A specialized editor is available via right-click on the block. This editor allows editing of the plain-text HTML, simultaneously displaying the formatted output. On save, the block dimensions are automatically updated to accommodate the formatted text. See https://docs.oracle.com/javase/tutorial/uiswing/comonents/html.html for a comprehensive discussion regarding the use of HTML in a component.

Properties:

Height	Height of the text block in screen units.
Text	Information to be displayed in the block.
Width	Width of the text block in screen units.

4.4.45 Or - Logic

Propagate the logical "or" of the inputs. This block expects multiple connections on its input port. It accomodates any number. The output is given a GOOD quality unless the block state is UNKNOWN. In that case the quality is set to the "worst" quality of any of the inputs. On a reset, existing values are retained, and an UNKNOWN value is assigned the output state. However, no new output is emitted until the next new value is received.

Properties:

SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 (no synchronization).

Connections:

in	incoming truth value.
out	truth-value, representing the “or” of the inputs. The output timestamp is updated

4.4.46 Output – Connectivity

Write data received on the input connection to an Ignition tag. Any type of incoming connection is allowed. The specified tag type must match the selected output connection type. Shortcut feature: If a tag is dragged from the tag browser and dropped on the right side of the diagram workspace it will become an Output block.

Properties:

TagPath	a fully qualified (i.e. includes provider name in brackets) path to a text tag

Connections:

in	incoming value.

4.4.47 Parameter – Connection

Achieve data persistence by encapsulating an Ignition tag. Incoming values are written to the tag. Tag changes are propagated on the output. There is no special reset behavior. Because Parameter blocks are associated with a tag, it is possible to have the same block replicated in several places in the collection of diagrams.

Properties:

TagPath	a fully qualified (i.e. includes provider name in brackets) path to a text tag

Connections:

in	incoming data value.
out	Data value, pass the input through the block.

4.4.48 Persistence Gate – Timer & Counters

The persistence gate guarantees that the input value has remained unchanged for a specified interval before that value is propagated. The block displays a count-down of the interval on its icon. When the block receives a value of good quality that matches the trigger property, it begins a countdown and passes the value when the countdown reaches 0. If the block receives a different value during the countdown, it aborts the countdown and passes the new value immediately. To specify how often the icon updates the countdown, set the ScanInterval attribute. The unit of time it displays is appropriate to the time remaining.

Properties:

ScanInterval	the interval between updates of the block ~ secs. The default value is 10. The scan interval must be at least 1 second.
TimeWindow	the length of time for which input values are monitored for no change ~ secs. The default value is 0 which effectively disables the block.
Trigger	the state (or ANY state) that starts the monitoring process.

Connections:

in	incoming truth value.
out	truth-value, as the behavior of the block dictates.

4.4.49 Product - Arithmetic

Compute the product of the inputs. This block expects multiple connections on its input port. It accommodates any number. On a reset, existing values are retained, and an UNKNOWN value is assigned the output state. However, no new output is emitted until the next new value is received.

What if any one of the inputs is unknown?

Properties:

SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 1 sec.

Connections:

in	Data, multiple.
out	double value representing the product of the inputs. The output timestamp is updated

4.4.50 Set Property - Control

When a new data value is detected on input, this block propagates a CONFIGURE signal on the output. This signal is universally understood by every block. If that block has a property whose name matches the argument parameter of the signal, then the value of the property in the block will be set to the payload of the signal. The block will be appropriately notified of the change. The signal will contain the name of the property (configured in the block), plus a value that is newly arrived on the input.

Check this…

Properties:

Property	the name of the property to be configured

Connections:

in	a data value. This value is sent to the output as a signal
out	a “configure” signal

4.4.51 Qualified Value - Misc

Create a qualified value from individual components: value, quality, and timestamp. The datatype of the output is the same as that of the value input. The input determines the default quality and timestamp as well. The block will emit an output for every value received on its input. The quality input is a string where “good” is interpreted as a good quality and all other value are bad. This sets the quality of the output. However, if the “value” input has bad quality, it will drive the output to a bad quality as well. Quality is taken as the last value received on the quality input. If no input has ever arrived on the quality channel, its contribution is considered to be good. The timestamp channel is either a date or a string input. If a string, its format is specified by the FORMAT property of the block. This is a string date-time format as specified by the Java SimpleDateFormat class. The default value is: “YYYY/MM/dd hh:mm:ss”. If the format does not include a date, the current date is assumed. If it does not include a time, the current time is assumed. The timestamp is taken as the last value received on the time input. If no input has ever arrived on the time channel, the timestamp of the output is assigned the timestamp of the value input.

A synchronization interval should be specified to guarantee proper pairing of readings between the separate input channels. The synchronization is driven by the “value” input only. This prevents re-issuance of a value on receipt of quality or timestamps only.

Properties:

Format	string format for the date-time. If the “time” input is textual (as opposed to a date) it is converted to a timestamp via this format.
SyncInterval	a “coalescing interval”. This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 1 sec.

Connections:

value	value of any data type. This forms the value component of the output
quality	string. If present, this forms the quality component of the output.
time	a date or a string in the format specified. If present, this forms the timestamp component of the output
out	qualified value constructed from the three inputs

4.4.52 Quotient - Arithmetic

Compute the quotient of two inputs. Report the value a port "a" divided by the value of the port labelled "b". On a reset, all existing values are forgotten.

Properties:

SyncInterval	a "coalescing interval". This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 1 sec.

Connections:

a	data value. The dividend
b	data value. The divisor
out	double value representing the quotient of the inputs. The output timestamp is updated.

4.4.53 Random - Statistics

This block is a noise generator. It alters the input value by selecting a random sample from one of several statistical distributions, adding it to the input and propagating the result. There is no reset behavior.

Properties:

Distribution	the random distribution, one of EXPONENTIAL, NORMAL or UNIFORM
Lower	least possible value when using a UNIFORM distribution
Upper	greatest possible value when a UNIFORM distribution is selected
Mean	when NORMAL distribution is used
StandardDeviation	when NORMAL distribution is used

Connections:

in	incoming data value.
out	double value representing the input perturbed by addition of a random sample selected from the specified distribution. The output timestamp retains the value of the input.

4.4.54 Range - Observation

Validate incoming input data values against a configured set of upper and lower limits with optional deadband ranges. The output is a truth-value according to the truth-table below. For values that are in ranges not specified, retain the most recent past value. On reset the past value is set to UNKNOWN.

TRUE	FALSE
X <= UpperLimit – UpperDeadband and X >= LowerLimit + LowerDeadband	X > UpperLimit or X < LowerLimit

Properties:

LowerLimit	the value of the lower limit, default is zero
UpperLimit	the value of the upper limit, default is zero.
LowerDeadband	tolerance around the lower limit, default is zero.
UpperDeadband	tolerance around the upper limit, default is zero

Connections:

in	data connection, the raw input.
out	truth-value connection, the result.

4.4.55 Readout - Misc

This block is a pass through that displays the current value of its input connection. The format must contain exactly one form of "%s" (for string data), or "%d" (for integer data), or "%f" (for floating point data). If no format is specified, "%s" is assumed. The format strings may be modified with a width and precision. E.g. "%5.3f" means "display leaving space for at least 5 digits with 3 following the decimal". Further options can be found in the Java 8 documentation (http://docs.oracle.com/javase/8/docs/api) for String.format().

Properties:

Format	the format in which to display the value
Value	the most recent value to pass through the block.

Connections:

in	data value.
out	data value. The timestamp is not altered from the input

4.4.56 Receiver - Connectivity

Accept a signal message from the "engine" and propagate it on the output connection. This block is largely superfluous since any block can be configured to receive signals.

Need some more explanation here, or a reference to a section that discusses receiver and transmitters and signals.

Properties:

Command	the command that the receiver will accept. All others are ignored.

Connections:

recv	data or truth value

4.4.57 Reset - Control

Propagate a reset signal to connected blocks when the input newly matches the trigger value or the block is explicitly evaluated.

Properties:

Trigger	TruthValue, the state required to trigger the signal.

Connections:

in	incoming truth value, this triggers the output if it matches the trigger value.
out	reset signal

4.4.58 Sink – Connectivity

Send values from the block to a memory tag for routing to Source blocks that are configured to receive from this block. Sink blocks share tag paths with Source blocks with which they are logically connected.

Properties:

TagPath	a fully qualified (i.e. includes provider name in brackets) path to a tag

Connections:

in	incoming value.

4.4.59 Source - Connectivity

Accept an input value from a memory tag and propagate it on the output connection. The tag is shared, presumably by a corresponding Sink block.

Properties:

TagPath	a fully qualified (i.e. includes provider name in brackets) path to a tag

Connections:

out	data connection, the tag value

4.4.60 SQC - Statistics

Execute one of the Western Electric/Westinghouse SPC rules on an input. A truth-value output records changes in the rule state. Any changes in either limit or target trigger a new rule computation. The rules are one-sided or two-sided x-out-of-y, or x consecutive points on one side of the limit.

An SQC block has three numeric input paths, the values carried over the paths are the target value (T), the standard deviation (S), and the actual measured value (V). The block has a single boolean output that is true if a SQC rule has been violated and false otherwise. Evaluation will take place whenever a new data value is presented along any one of the three paths (there is no special input synchronization because, normally, the target and standard deviation values are fairly static). All three data paths must have GOOD data quality. If not, the output result is UNKNOWN, with a bad quality.

As data points are propagated to the block on the data path connected to the (V) input port they are stored in a buffer that is sized based on the sampleSize property. The algorithm checks the number of data points in the block's history as follows:

- If the number of out-of-range points equals or exceeds the limit then the output is TRUE.
- If the number of out-of-range points is less than the limit and there are insufficient unfilled slots in the data buffer to reach that limit, then the output is FALSE.
- If neither of the above conditions hold, the output is UNKNOWN.

On reset(),if the clearOnReset property is TRUE, the buffer is cleared and the block reports an UNKNOWN status. NOTE: the block does not poll for data. It only detects changed values on its input.

SQC blocks communicate status. If a one-sided SQC rule has been violated, then the block that detected the violation notifies all remaining SQC blocks on the same diagram. Any one-sided SQC rules on the opposing side must then report FALSE, if currently TRUE. This extra-ordinary block report has no effect on the current state of the block, or its history buffer. Two-sided blocks neither initiate nor participate in this capability. This is important so that the most recent SQC violation is the one that is driving the recommendation and corrective action. The signals that are propagated are: CLEAR_LOWER_SQC and CLEAR_UPPER_SQC.

Calculate the control limits as follows. The control limits rarely change but could change between block executions.

Upper = target + standard-deviation * no.-of-standard-deviations-to-test

Lower = target - standard-deviation * no.-of-standard-deviations-to-test

Count the number of data values violating the control limits. Compare to the SQC rules.

Properties:

ClearOnReset	If TRUE, the value buffer will clear when the block is reset. By default this property is FALSE.
NumberOfStandardDeviations	the distance above or below the target which signals a violation. This is expressed in standard deviations. This parameter is used for HIGH, LOW or BOTH limit types.
LimitType	CONSECUTIVE, HIGH, LOW or BOTH
MinimumOutOfRange	minimum number of points exceeding the limit within the stored history in order for a TRUE value to be concluded. In the case of CONSECUTIVE this value refers to the minimum number of consecutive points on the same side of the limit to trigger a violation.
Mean	the nominal process mean.
SampleSize	number of observations to retain for computation of rule state.

Connections:

T	target value.
S	standard deviation
V	the sample value
out	truth-value, result of evaluation of the inputs, initialized to UNKNOWN.

4.4.61 SQC Diagnosis - Conclusion

Block with a description that can be used by downstream blocks to interrogate the reason for this diagnosis. It will write its status to a tag, if a tagpath is provided. One important use of SQC Diagnosis is as an entry point for the SQC plotting utility. There is a configuration dialog to set the label that can be accessed by downstream blocks.

Doesn't this have a description or explanation property?

Properties:

TagPath	a fully qualified (i.e. includes provider name in brackets) path to a boolean tag
Label	Used for SQC plotting

Connections:

in	incoming truth value indicating a diagnosis condition.
out	truth-value, pass the input through the block.

4.4.62 State Lookup - Control

This block has a custom configuration dialog that allows the specification of an arbitrary number of case-insensitive state names. Each state is associated with a truth-value to be propagated on the output. The special state "OTHER" matches any input that does not correspond to any of the fixed values.

Is there a popup configuration or just a list property with values?

Properties:

SyncInterval	a “coalescing interval”. This is the time allowance for multiple values that have originated in the same scan to be evaluated at the same time. ~ secs. The default value is 0 (no synchronization).
NTrue	the number of inputs which must carry a “true” value in order to propagate a “true” on the output. The default value is 0 (always succeeds). *What does this do?*

Connections:

in	string value, a parameter, the state name to be evaluated
out	truth-value, representing the state of the corresponding match.

4.4.63 Statistics – Statistics

There are three blocks that calculate selected statistical values. The actual metric is selectable for each block from the following choices: geometric-mean, mean (arithmetic), median, maximum, minimum, range, standard deviation, variance.

4.4.63.1 Statistics - Statistics

Apply the selected statistical function to the set of latest values from each of the connected inputs.

Properties:

ClearOnReset	If TRUE, the computation will re-start, requiring new inputs from each connection, when the block is reset. By default this property is FALSE.
Function	The name of the statistical function to apply.

Connections:

in	incoming data values. This anchor point allows multiple connections.
out	data, the calculated statistic.

4.4.63.2 Statistics (n) - Statistics

Compute the specified statistic on specified number of the most recent values that have arrived on the block’s input. Once the buffer is filled, report the statistic on the arrival of each new point. Values of bad quality are ignored for calculation purposes. When bad value is received, then the output is given a bad quality. The result may, optionally, be cleared on reset.

Properties:

SampleSize	the number of points to include in the calculation. When the sample size is changed, the buffer is cleared
Function	The name of the statistical function to apply.

Connections:

in	incoming data value.
out	data, the rolling average.

4.4.63.3 Statistics (t) - Statistics

Compute the specified statistic on a fixed-interval sampling of values that have arrived on the block's input within a specified time window. Report results on the sample interval. Values of bad quality are ignored for calculation purposes. When a bad value is received, then the output is given a bad quality. The result may, optionally, be cleared on reset.

Properties:

ClearOnReset	If TRUE, the average computation will re-start when the block is reset. By default this property is FALSE
ScanInterval	the sampling interval ~ secs
TimeWindow	the period over which the calculation is to take place. ~ secs
Function	The name of the statistical function to apply.

Connections:

in	incoming data value.
out	data, the accumulated average

4.4.64 Sub Diagnosis - Conclusion

Block with a description that can be used by downstream blocks to interrogate the reason for this diagnosis. It will write its status to a tag, if a tagpath is provided. One important use of SQC Diagnosis is as an entry point for the SQC plotting utility. There is a configuration dialog to set the label that can be accessed by downstream blocks.

Doesn't this have a description or explanation property?

Properties:

Label	a text string useful to downstream blocks to interrogate for diagnosis condition information
TagPath	a fully qualified (i.e. includes provider name in brackets) path to a boolean tag

Connections:

in	incoming truth value.
out	truth-value, pass the input through the block.

4.4.65 Time Readout - Misc

This block has no behavior. It simply propagates any value on its input to its output. Its function is to provide a place to display the time of the last value that passed through.

The format must be a valid SimpleDateFormat. If no format is specified, "YYYY/MM/dd hh:mm:ss" is assumed. Further options can be found in the the Java 8 documentation (http://docs.oracle.com/javase/8/docs/api) for SimpleDateFormat.

Properties:

Format	the format in which to display the time. See java documentation for SimpleDateFormat for format configuration
Value	the most recent value to pass through the block.

Connections:

in	Any data type.
out	Pass through of the input data value.

4.4.66 Time Fork - Misc

Take the timestamp of the incoming value and pass it on the output.

Connections:

in	a data value of any type
out	the timestamp of the input

4.4.67 Timer – Timers & Counters

When the triggering value is received on the input, this block starts a counter recording the number of seconds in the triggering state. The counter is evaluated at the configured interval. The trigger may be any TruthValue. The StopOn value may not be the same as the trigger. On block reset, the state is changed to UNSET. This does not occur if a RESET

signal is received. On reset, the counter is cleared (or not depending on the state of AccumulateValues). If, at the time of reset, the block state matches the trigger, then the count cycle resumes. Otherwise a single value of zero is propagated. If a tag path is configured, this block saves the count value in the tag at the same time it is updated, thus making it available for other applications.

Properties:

AccumulateValues	if TRUE, the timer value is NOT reset when the block returns to its triggering value. Otherwise the count is zeroed on start of "on" interval.
Interval	the sampling time, ~seconds. Output is generated at this rate. The default value is 1 second.
TagPath	a fully qualified (i.e. includes provider name in brackets) path to a integer tag. If configured, this tag will receive the counter value at the evaluation interval.
Trigger	a TruthValue. The value to be matched on input in order to trigger the counter. Default value is TRUE.
StopOn	the value to be matched on input in order to stop the counter. Default value is FALSE.

Connections:

in	a truth value. This triggers the counter if it matches the trigger value
out	the counter value

4.4.68 Trend Detector - Statistics

The Trend Detection block has the same inputs as an SQC block – target, standard deviation and value. It implements trend detection using a least squares regression algorithm. The evaluator will fire if a new data value is propagated along any one of the three input paths as long as there is good data on each of the three. In addition to providing a Boolean output signaling that it has detected a trend, the actual slope, slope variance, and projected value are also available as outputs of the block and may feed other blocks if needed. On reset, the block optionally clears its local history.

Properties:

ClearOnReset	If TRUE, the value buffer will clear when the block is reset. By default this property is FALSE
NumberOfTrendPointsRequired	number of observations to retain for trend computation. This is the buffer size.
RelativeToTarget	if TRUE, detection is relative to the target value, otherwise it is relative to the actual mean.
SlopeCalculationOption	AVERAGE, or LINEARFIT
StandardDeviationMultiplicativeFactor	the incoming standard deviation is enhanced by this factor.
TestLabel	the trend test label.
TrendCountThreshold	number of observations in one direction or the other required to compute a trend
TrendDirection	DOWNWARD, UPWARD, or BOTH

Connections:

T	target value.
S	standard deviation
V	the sample value
out	truth-value, result of evaluation of the inputs, initialized to UNKNOWN.
slope	when a trend is detected this output propagates the computed slope
variance	when a trend is detected this output propagates the computed standard deviation. (The port name was chosen to avoid confusion with “slope” in the block portrayal)
projection	when a trend is detected this output propagates the computed value

4.4.69 Transmitter - Connectivity

Broadcast the signal received on the block input. This block is largely superfluous since any block can be configured to transmit.

Properties:

Scope	a scope which determines to which blocks the signal will be delivered.

Connections:

send	signal

4.4.70 Truth Counter – Timers & Counters

Count the number of times a given truth value has been received on the block input. Consecutive identical values are ignored. On RESET, set the count to a specified initial value.

Properties:

InitialValue	the initial count and count setting after a RESET.
Trigger	the truth value that is being counted. Default value is TRUE.

Connections:

in	incoming truth value.
out	the count, an integer

4.4.71 Truth Pulse - Connectivity

On initial start of a diagram, a block (or diagram) reset, or when the signal connection receives a RESET, this block sets its output momentarily to the configured value, waits for specified interval and then returns its output to the opposite of the pulsed value.

Properties:

Interval	the length of time to hold the output, ~seconds. The default value is 1 second.
PulseValue	the value to be propagated during the pulsing phase. Default value is TRUE

Connections:

out	a momentary TruthValue value (TRUE or FALSE)

4.4.72 Unknown - Logic

This block emits a TRUE if its incoming value is UNKNOWN.

Connections:

in	incoming truth value.
out	truth-value, TRUE if its incoming value is UNKNOWN. The timestamp is not altered from the input.

4.4.73 XY Fit - Arithmetic

Compute the least squares fit from a specified number of the most recent values that have arrived on the block's 2 inputs. Once sufficient values (2) have been received, report the

best fit value for the most recent arrival on the “fit” port using the least squares calculation. The scaled slope is reported on the “slope” port.

Properties:

ClearOnReset	If TRUE, the linear fit calculations will re-start when the block is reset. By default this property is FALSE
FillRequired	If TRUE, the linear fit calculation does not take place until the number of points determined by the sample size. Otherwise only 2 points are required
SampleSize	the number of points to include in the calculation. When the sample size is changed, the buffer is cleared.

Connections:

in	data, the raw input
out	best fit value for the current input
slope	slope over the current history, multiplied by the scale factor

4.4.74 Zero Crossing - Observation

This block emits a TRUE whenever the input value goes from positive to negative or negative to positive.

Connections:

in	incoming numeric value.
out	truth-value, TRUE if the sign of the incoming value is different from the previous value. The timestamp is not altered from the input.

4.5 Bad Data Handling

The information transmitted between blocks is encapsulated in QualifiedValue objects. These contain a value, a quality and a timestamp. In general, when a data value of bad quality is received, the resulting output will also be marked as having bad quality. In addition, the result of any calculation involving the bad data will return a value of not-a-number or UNKNOWN as appropriate to its type.

Exceptions to this pattern occur when it is possible to deduce an answer without involving the spurious input. For example, an “And” block can conclude a FALSE with one FALSE input of good quality, independent of the value or quality of other inputs.

4.6 Block Locking

Any block may be placed in a locked state. When locked, no further results are transmitted on its output, however all input processing occurs as normal. When the block

becomes unlocked, output is again enabled. The block will output a value during its next normal evaluation. Depending on the block this may be as a result of a timer expiring or receipt of input.

The locked state is activated via a menu choice that is available on all blocks. A locked block is annotated with a lock badge in the designer. Locked state is always cleared during a save.

Even when locked, the block will respond to a “forcedPost” request,. This causes the block to place a specified value on its output. This allows a tester/developer to isolate portions of a diagram during development/debugging.

4.7 Block Reset

All blocks support a reset() command. The intent of a reset() is to clear the block state and any internal data structures, thereby forcing calculations to begin from scratch. A reset will place the block in an UNSET state. This state is not propagated downstream. However, the downstream connection will be painted in such a way as to mark the reset action.

The UNSET state guarantees that the receipt of the next value will cause the block to evaluate and propagate a value, if appropriate. Blocks, such as Compare, And or Or, (these accept multiple inputs) retain the latest values received on each input. This allows a computation to be made on the very next arrival, no matter which input.

On a diagram reset, Input blocks propagate their values (the value of the bound tag). Individual blocks in the diagram are reset first before any propagation from input blocks.

4.8 Timestamp Handling

In general, if a data value is passed through a block, even if there has been an operation on it (e.g. exponential), the timestamp of the result will be the same as the timestamp of the input. However, if the value has been combined in any way with other inputs, then the timestamp of the result is set to the time at which the output was propagated.

4.9 Connection Type Change

Several of the block classes operate on any of the possible data types. In order to properly configure connections for each individual situation, a control-click menu option is supplied to “Change Connection Types”. This option alters all of the block’s stubs to the chosen type. The blocks for which this feature is available include:

Input

Junction

Output

Parameter

Readout

Sink

Source

Once a diagram has been saved, this option is no longer available.

4.10Type Coercion

Blocks are aware of the types of data they process. The type of the input connector guarantees the data type of an input. (In the case of Input blocks, the incoming tag value is coerced to the expected type). Blocks also guarantee the correct data type on output.

Where coercion is necessary, the following steps are taken:

The value is converted to a string. This provides the conversions with a common starting point.

For strings no conversion is necessary. However, a null will generate an error.

For real numbers, the conversion is Double.parseDouble(value).

For integers, the conversion is Integer.parseInt(value).

For booleans, any non-zero numeric value or the string “TRUE” (case-insensitive) is taken as true. All others are false.

If the conversion step fails, then the reading is given a BAD quality and propagated.

4.11Creating Custom Bocks

It is possible for users to create custom blocks without requiring a rebuild of the BLT module. Several of the blocks that are distributed with Symbolic Ai have been written in Python. It is transparent to the user whether a block is implemented in Java or in Python. The python blocks that have been written by ILS Automation are distributed in the installer and placed into the `user-lib/pylib/ils/block` subdirectory of the Ignition install location. These Python files will be distributed with every Symbolic Ai release so any changes made to these files will be lost. Python for user defined blocks should be placed into the `user-lib/pylib/ils/user/block` subdirectory of the Ignition install location. These files will not be overwritten. New blocks placed into either of these locations will automatically recognized upon a Gateway restart.

The following blocks are implemented in Python and can be reviewed and used as a pattern for new blocks:

- Action - ils//block/action.py
- Arithmetic - ils/block/arithmetic.py
- FinalDiagnosis - ils/block/finaldiagnosis.py
- SQCDiagnosis - ils/block/sqcdiagnosis.py
- SubDiagnosis - ils/block/subdiagnosis.py

4.11.1 Block Icons

Each block needs to provide two icon images; one for the palette toolbar and another for the design workspace. They should be 32x32 and 100x100 PNG files, respectively, for best results. Image files are loaded into Ignition via the Image Management system in Designer.

Note: ILS uses the "Paint S" package on a Macintosh to create and edit images. After the image editing is complete the paint file is saved as a PNG and loaded into Ignition.

4.11.2 Init File Requirement

New block names need to be added to the __init.py__ file located in the same folder as the block python file. The new block will not appear in the palette if this entry is not added. Please note that the name and file name must match and be all lower case in order for it to be picked up by the system. The __init.py__ entry should be similar to this from the `ils/block` folder:

```
__all__ = ["action", "arithmetic", "finaldiagnosis",
      "sqcdiagnosis", "subdiagnosis", "mynewblock"]
```

4.11.3 Block Properties

Each block possesses a pre-defined list of properties. Properties are persistent attributes specified by a name, a binding type and a value. Additionally, a property value may be configured for display in the designer by using a right mouse click on a block and selecting Display Block Properties. This will bring up a dialog box where available properties can be selected for display. Property values do NOT have a quality attribute.

4.11.3.1 Data Types

Properties are pre-assigned a data type from one of the following options:

BOOLEAN

INTEGER

HTML (gets an HTML editor)

HYSTERESIS (TRUE, FALSE, ALWAYS, NEVER)

LIMIT (HIGH, LOW, BOTH, NONE, CONSECUTIVE)

LIST (multiple user-entered values)

OBJECT (matches any type)

OPTION (block-defined selection list)

SCOPE (GLOBAL, APPLICATION, FAMILY, LOCAL)

STRING

TIME

TRUTHVALUE (TRUE, FALSE, UNKNOWN, UNSET)

4.11.3.2 Binding

Block properties may be bound to tags or other elements in the application. The binding options are as follows. Except for properties that are pre-defined as bound to the ENGINE or OPTION, the binding definition can be changed dynamically via the property editor.

Binding	Description
NONE	
ENGINE	The ENGINE designation is determined exclusively by the block and cannot be changed. That is, once a property is designated as ENGINE it cannot be set to a different binding type, nor can a property that is not bound to ENGINE be converted to ENGINE. This behavior is enforced by the property editor
OPTION	In a similar way, the OPTION binding contains a fixed list of options known to the block and cannot be changed
TAG_MONITOR	The TAG_MONITOR binding describes a property that subscribes to a tag and changes its value when the tag changes value. An Input block is the most straightforward example
TAG_READ	A property bound to a tag with TAG_READ actively retrieves the tag value on command
TAG_READWRITE	TAG_READWRITE is used for a property that both writes to a tag on value change and refreshes its value when the tag changes. This is used, for example, in the Parameter block. The block is essentially a stand-in for the tag in the diagram
TAG_WRITE	TAG_WRITE refers to a property binding where its value is written to a tag each time it changes

The value of any property can, optionally, be displayed in the Designer by using the Display Block Properties option from the block's right click popup menu. Properties that have an ENGINE binding are of particular interest as they reflect some internal state of the block and will update dynamically as the block executes.

4.11.3.3 Activity Log

Every block has the potential to record a list of activities in a fixed-size buffer. Initially this capability is disabled. It can be activated simply by editing the block's *ActivityBufferSize* property, setting the value to something greater than zero.

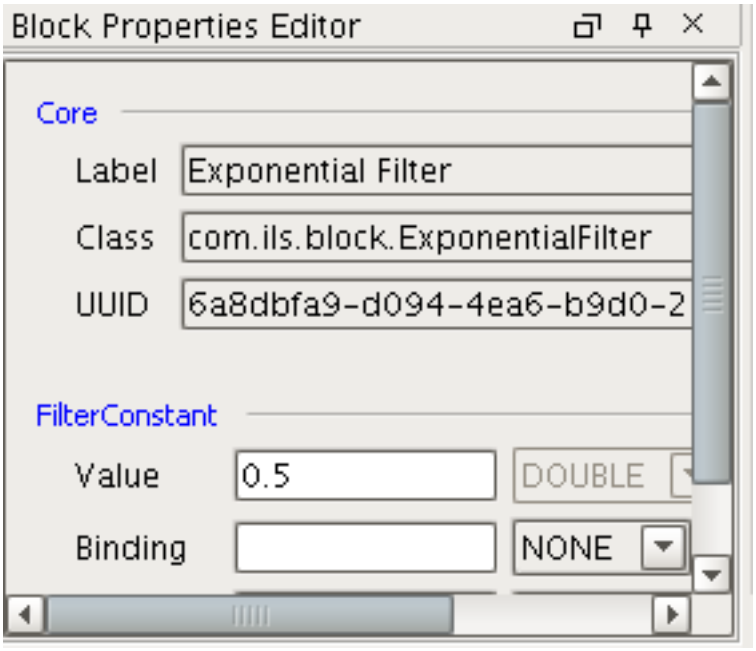
Contents of the log are available in the "View Internal State" popup menu on the block. Activities that are logged include:

- RESET – marks a reset action on the block
- STATE – records an internal block state change
- port – records data propagated on the named output port
- Additional activities may be available on specific classes of blocks.

4.11.3.4 Property Editor

Whenever a block is selected, an editor for its properties appears in a Designer (Wicket) window. The editor displays read-only block attributes, and then a panel for each custom property. When edited, the block properties are immediately transmitted to the Gateway.

Note: Entries into a text field are not recorded until an ENTER is typed.

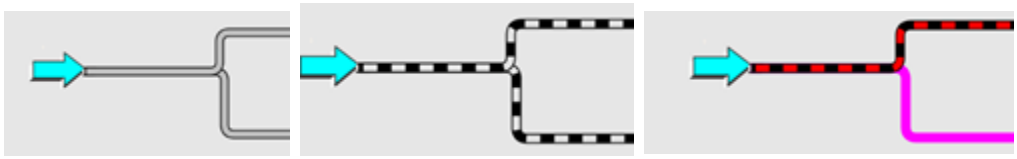


4.11.3.5 Connections

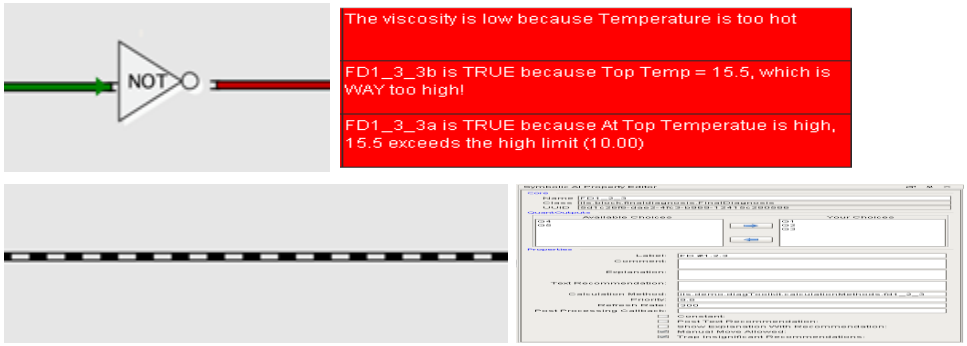
There are four types of connections:

- Numerical (data)
- Truth-value (true,false,unknown,unset)
- Signal

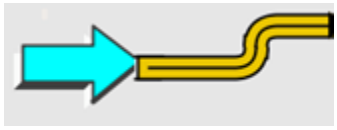
_ Text (diagnoses, recommendations)



DATA – normal, unset, not-a-number, selected (magenta)



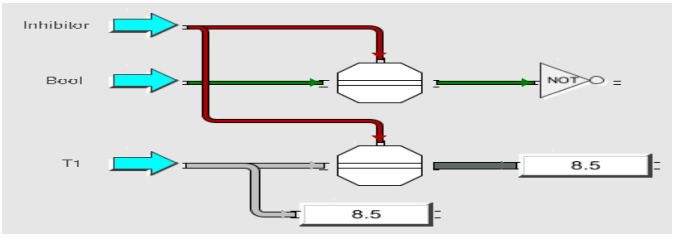
TRUTH-VALUE – true (green), false (red), bad quality(yellow/black), unset (gray/black), unknown (gray)



TEXT

4.11.3.6 Transmit/Receive

The ability to send and receive signals is a design-time configurable property of a block. Blocks enabled to receive broadcast signals are marked with a receiver “badge”. Blocks that send signals are similarly marked with a transmitter badge.



4.11.3.7 Signals

Signals are commands that can be used to control block state, e.g. “start” or “reset”. In addition to originating from blocks within a diagram, signals can be sent from the outside, e.g. from a Vision window. Signals have a property of “scope” that determines the collection of blocks to which they may be delivered.

There are four scoping levels:

- Local – the signal is delivered only blocks in the diagram from which it originated
- Family – the signal is delivered blocks in diagrams in the same family as the originator
- Application – the signal is delivered to blocks in diagrams in the same application as the originator
- Global – the signal is delivered to all blocks (with receiving enabled) in the current project

A signal has the following attributes:

- Command – a string value containing a well-known command
- Argument - a string argument or parameter name
- Payload – an optional payload, a string
- Pattern – a string used by a Receiver block to determine whether or not to process the signal. A '*' configured in the block matches all input. For other types of blocks, the pattern must match the block name.

Standard Signals - All blocks are expected to respond to the following signal commands as they arrive on the signal input stub that is present for every block.

- configure – use information in the signal to set a property value. Trigger any evaluations would normally occur.
- propagate – execute the propagate() method of the block. This forces emission of the last value on the outputs.
- lock – set the block to a locked state. It will not pass any values.
- reset – clear any internal storage, set the current state, if appropriate, to UNKNOWN.
- unlock – set the block to an unlocked state

4.11.4 Sample User Block

A sample block file (sample.py) is provided to help users understand the required methods in a block definition; it references images files that don't exist in order to keep the sample block from appearing on the palette tab. Block python files contain a dictionary of attributes that define how the block will appear on the palette and diagram:

```
def getPrototype(self):
    proto = {}
    proto['iconPath']= "Block/icons/palette/sample.png"
    proto['viewIcon']      = "Block/icons/embedded/sample.png"
    proto['label']   = "Sample"
```

```
proto['tooltip']        = "Sample user created block"
proto['tabName']        = 'Logic'
proto['viewBackgroundColor'] = '0xF0F0F0'
proto['blockClass']     = self.getClassName()
proto['blockStyle']     = 'square'
proto['viewHeight']     = 70
proto['viewWidth']      = 70
proto['inports']        = self.getInputPorts()
proto['outports']       = self.getOutputPorts()
proto['receiveEnabled']  = 'false'
proto['transmitEnabled'] = 'false'
return proto
```

The primary method of concern is acceptValue. This method is called whenever new data appears on an input port. Several other methods are available to override in the block classes for special circumstances. Please refer to ils/actions/block/basicblock.py for a comprehensive list of available methods.

4.12Signal Processing

The toolkit provides the ability to send a signal to a block. A signal is analogous to making a menu selection when clicking on a block but allows it to be automated or added to some external logic. The following signals are currently defined:

- Reset – this is the most common signal to send and is supported by many blocks
- Evaluate
- Inhibit – only appropriate for the *Inhibitor* block

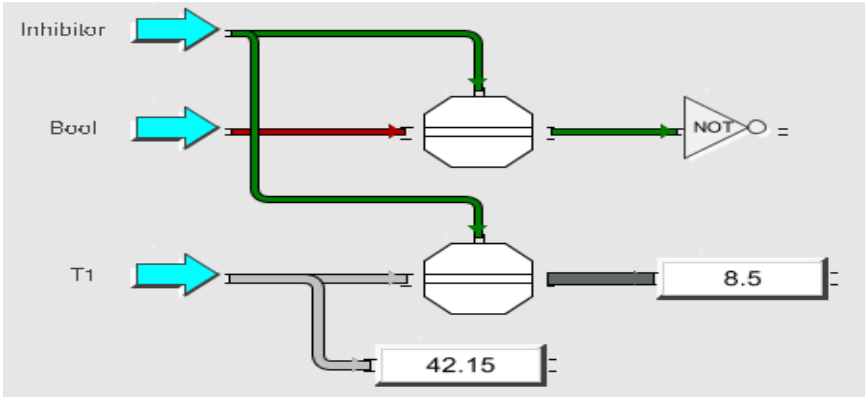
There are two blocks that can send a signal: the reset block and the command block. The command block is configured with the signal to send.

There are three ways to send a signal, each is described in the following paragraphs:

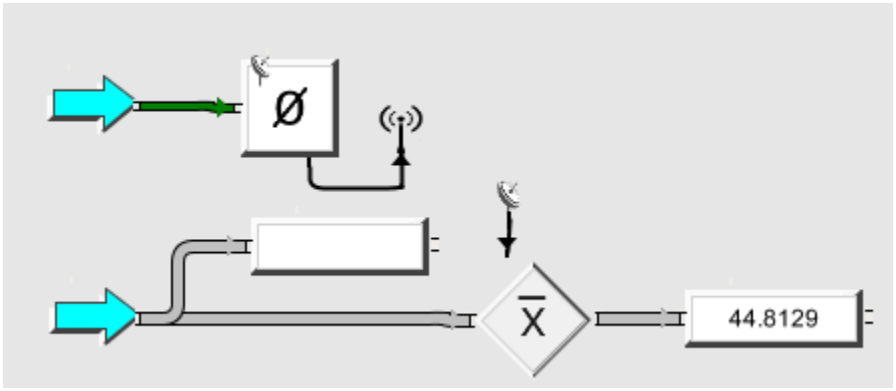
- Via a signal connection
- Via the transmitter and receiver
- Programmatically

The diagram below shows the reset signal from a reset block sent to a Moving Average block via a signal path. The signal path is a thin solid connection. Because signal paths are an advanced feature that are not terribly common, blocks do not normally show the

signal connection. Blocks that support a signal connection include a menu choice *Show Signal Connection*.



The diagram below shows the reset signal from a reset block sent to a Moving Average block via a transmit / receive block.



The Python snippet below shows how to send a inhibit signal programmatically. In this case the diagramUUID and blockName comes from a Vision window but in a typical example it will come via an action step or Final diagnosis callback.

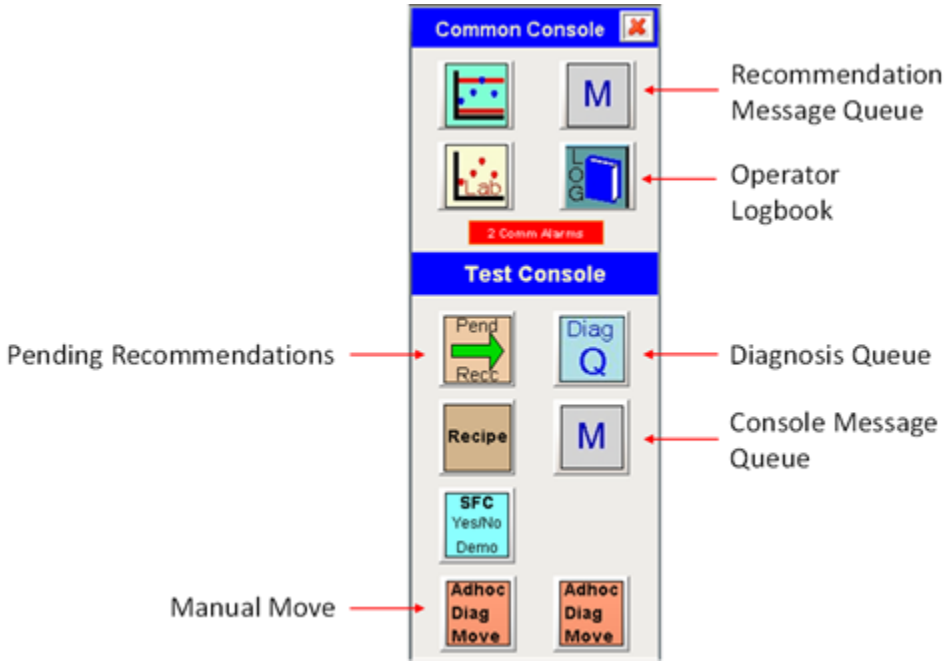
```
import system

diagramUUID = event.source.parent.getComponent('Diagram UUID Field').text
blockName = event.source.parent.getComponent('Data Inhibitor Field').text

system.ils.blt.diagram.sendSignal(diagramUUID, blockName, "inhibit", "Testing")
```

5 Client User Interface

There are several screens used by the operator.

5.1 Operator Console

The console window has a number of buttons with interesting icons that allow the operator quick access to various facilities. The ones that are relevant for the diagnostic toolkit are shown and described below.

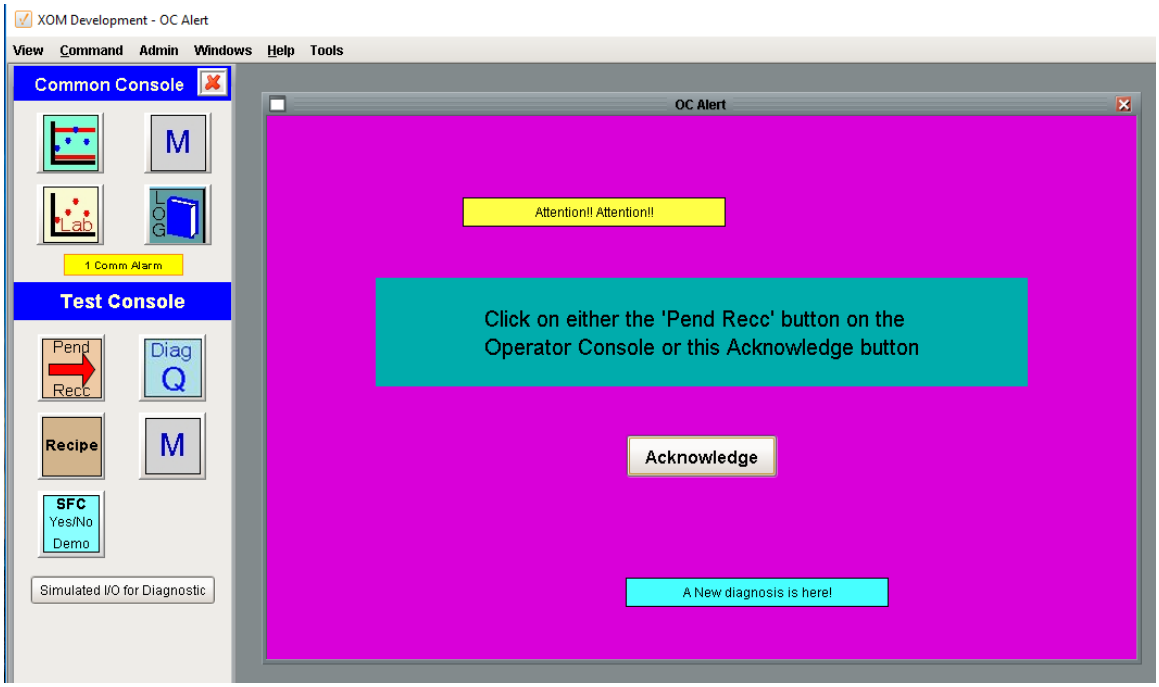


Button	Description
Recommendation Message Queue	Displays a queue of recommendation messages. Refer to section 2.4
Operator Logbook	Records all significant actions taken by the operator including the actions taken from the setpoint spreadsheet and the status and value of all attempted writes.
Pending Recommendations	The arrow indicates if there is an active recommendation. When green there are no active recommendations. When red there is an active recommendation. Pressing the button will display the setpoint spreadsheet if there is a numeric recommendation or a modal dialog if there is a textual recommendation
Diagnosis Queue	Displays a queue of diagnosis that have been made, including the state of the diagnosis and the recommendation associated with the diagnosis. Refer to section 3.9
Console Message Queue	Refer to section 2.4
Manual Move	Refer to section 3.6

Refer to section 7.2 for details about the various queues. Additionally, there is a queue for each application which is only available from the **Queue -> View** main menu.

5.2 Notification of New Recommendation

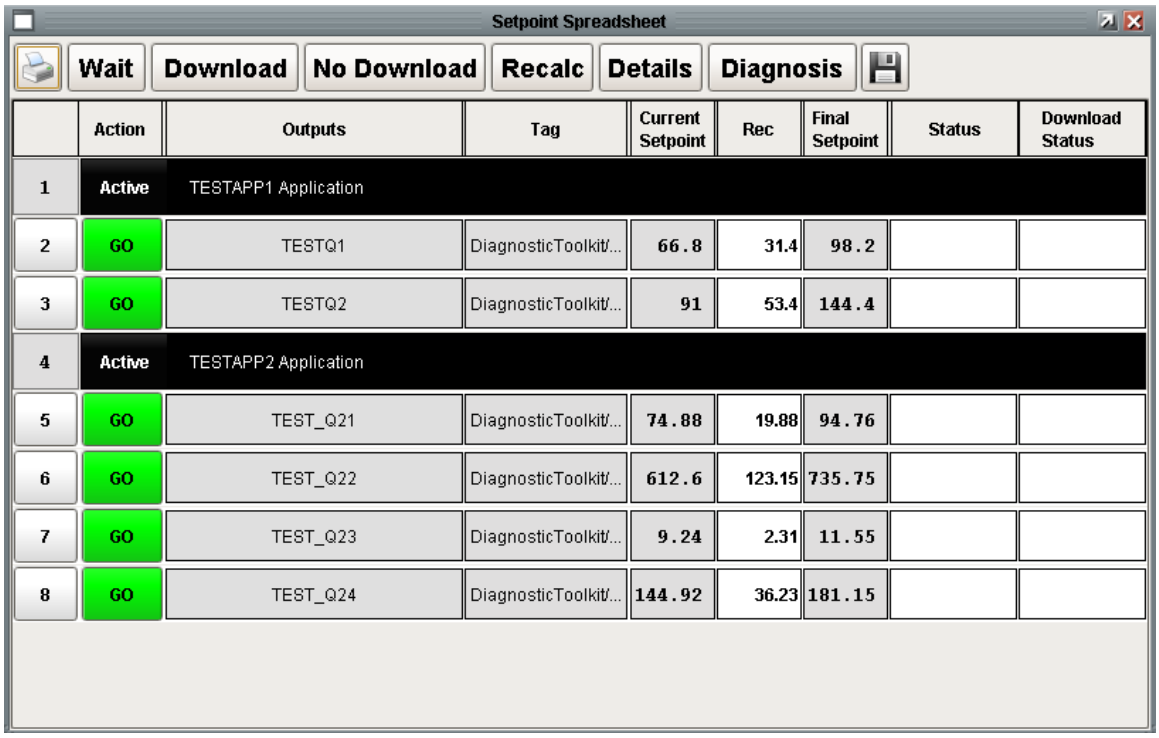
The diagnostic toolkit processing occurs in the gateway. When a problem is detected and a diagnostic recommendation for new setpoints is made, the appropriate clients are notified using the common OC alert. The OC alert is difficult to miss, however, if the client connects after the notification is sent then the "Pend Recc" icon on the console window will be animated with a red arrow. Pressing the "Acknowledge" button or on the "Pend Recc" button will open the setpoint spreadsheet.



5.3 Setpoint Spreadsheet

The setpoint spreadsheet is shown below. It displays the recommendation from the diagnostic toolkit for all of the highest priority problems that are active for each application. The setpoint spreadsheet allows the operator to:

- Download the recommendations to the DCS
- Selectively ignore individual outputs and/or recommendations
- Manually edit the recommendation
- Recalculate the recommendations to use the latest real-time data
- View the details of an individual output via the recommendation map window.



	Action	Outputs	Tag	Current Setpoint	Rec	Final Setpoint	Status	Download Status
1	Active	TESTAPP1 Application						
2	GO	TESTQ1	DiagnosticToolkit/...	66.8	31.4	98.2		
3	GO	TESTQ2	DiagnosticToolkit/...	91	53.4	144.4		
4	Active	TESTAPP2 Application						
5	GO	TEST_Q21	DiagnosticToolkit/...	74.88	19.88	94.76		
6	GO	TEST_Q22	DiagnosticToolkit/...	612.6	123.15	735.75		
7	GO	TEST_Q23	DiagnosticToolkit/...	9.24	2.31	11.55		
8	GO	TEST_Q24	DiagnosticToolkit/...	144.92	36.23	181.15		

5.3.1 Absolute Recommendations

Regardless of whether a quant output is defined as incremental or absolute, the setpoint spreadsheet always shows the current setpoint, the incremental change, and the final setpoint. An absolute quant output is generally used when a fixed setpoint is desired regardless of what the current setpoint is.

Absolute recommendations are subject to the same clamping constraints as incremental recommendations.

5.3.2 Recalculating Recommendations

There are two means of recalculating recommendations, manual and automatic. Both methods rerun the calculation methods of the active final diagnosis and read in fresh setpoints. Both methods overwrite any manual edits that may have been entered.

5.3.2.1 Manual Recalculation

The operator is provided a "Recalc" button on the setpoint spreadsheet that will run the calculation methods again and update the recommendations. This button is often referred to as the "Oh Shit" button because it will completely overwrite any manual entries that have been made, both through the setpoint spreadsheet or the recommendation map. It is a convenient way to get back to the original, more precisely, updated versions of the original, if they are not comfortable with the manual changes they have made. It is also an on demand way to get updated recommendations if the setpoint spreadsheet has been open for an extended period of time.

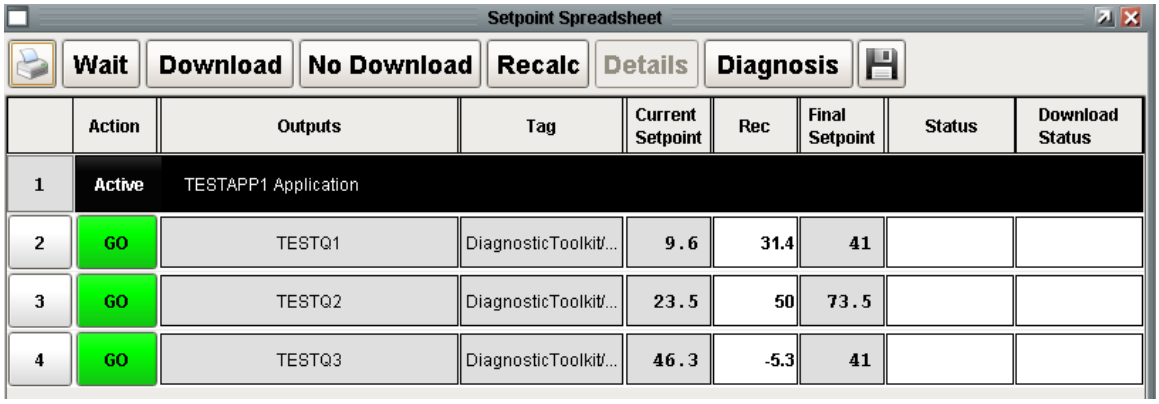
5.3.2.2 Automatic Recalculation

An automatic recalculation as determined by the *Refresh Rate* of a Final Diagnosis. The *Refresh Rate* is generally a relatively large number, the default is 5 minutes, so it only is a factor when the operator is busy with other activity and not paying attention to the setpoint spreadsheet.

5.3.3 Output Clamping

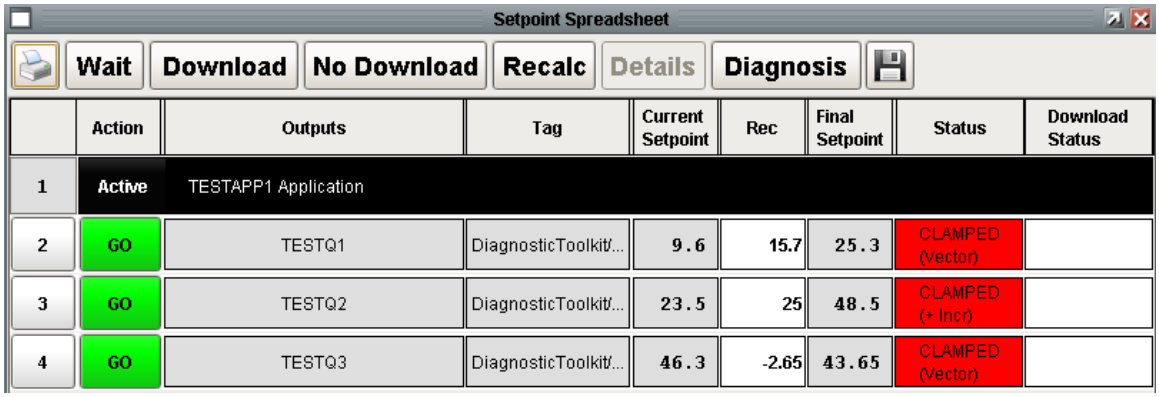
Output clamping is used to enforce the output constraints which were defined in section 3.2. Output clamping is best explained by an example. Consider a problem which is

corrected by making the following three recommendations with output constraints that are much larger than the recommendations (test12aa).



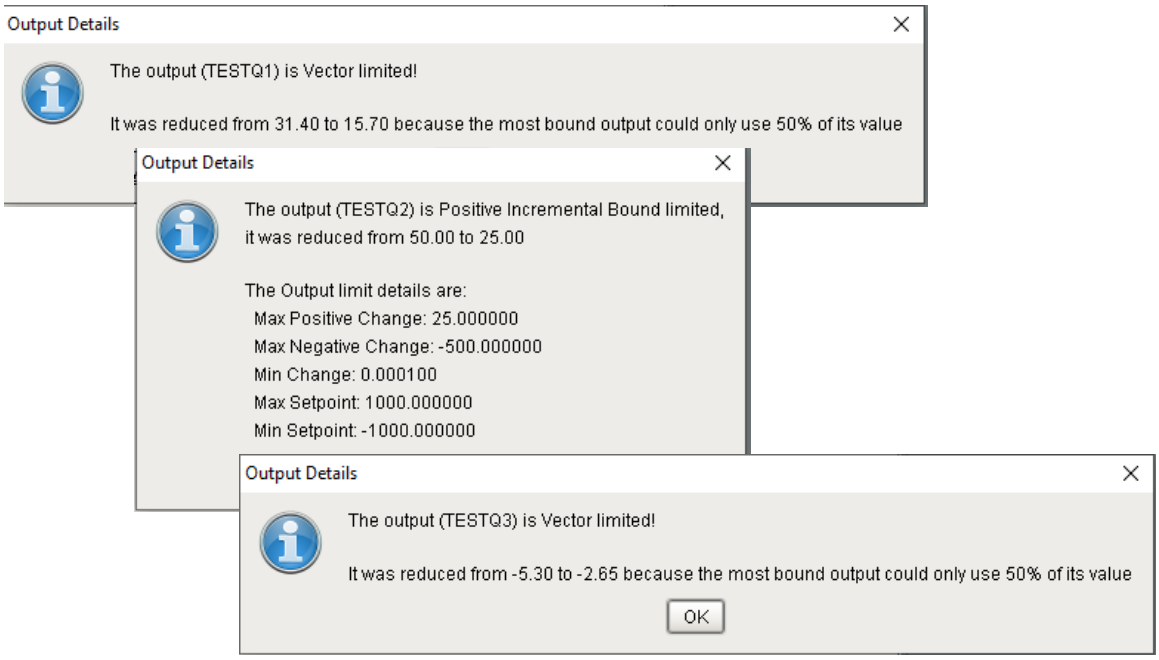
	Action	Outputs	Tag	Current Setpoint	Rec	Final Setpoint	Status	Download Status
1	Active	TESTAPP1 Application						
2	GO	TESTQ1	DiagnosticToolkit/...	9.6	31.4	41		
3	GO	TESTQ2	DiagnosticToolkit/...	23.5	50	73.5		
4	GO	TESTQ3	DiagnosticToolkit/...	46.3	-5.3	41		

If vector clamping is enabled and after defining a maximum positive increment of 25 for TESTQ2 and running the same calculations, we get the following (test12a):

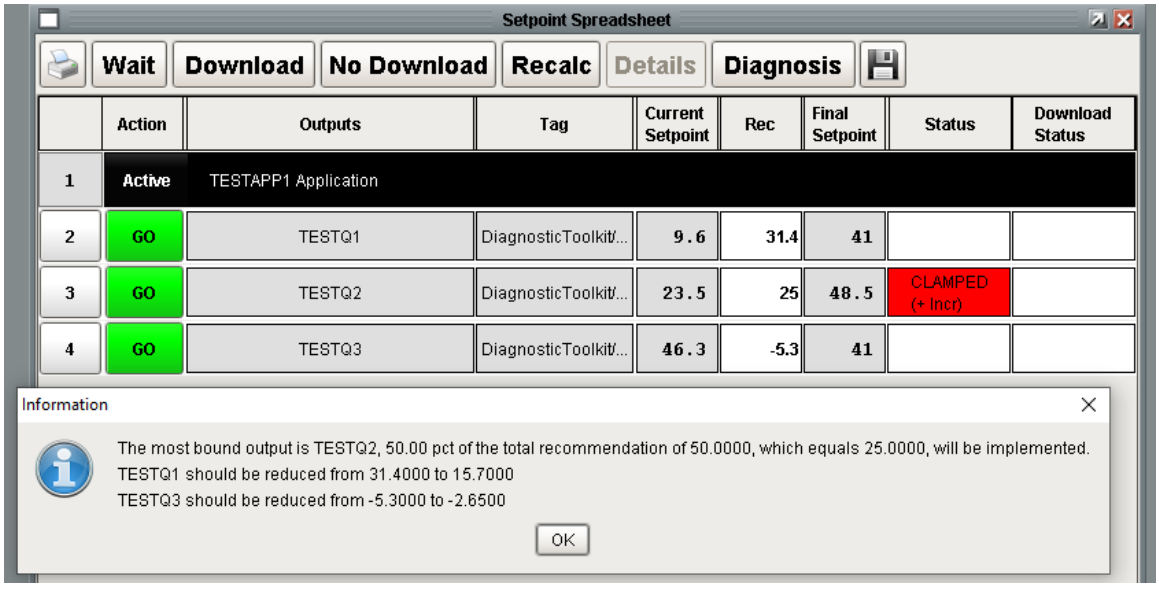


	Action	Outputs	Tag	Current Setpoint	Rec	Final Setpoint	Status	Download Status
1	Active	TESTAPP1 Application						
2	GO	TESTQ1	DiagnosticToolkit/...	9.6	15.7	25.3	CLAMPED (Vector)	
3	GO	TESTQ2	DiagnosticToolkit/...	23.5	25	48.5	CLAMPED (+ Incr)	
4	GO	TESTQ3	DiagnosticToolkit/...	46.3	-2.65	43.65	CLAMPED (Vector)	

All 3 outputs are now clamped even though the output constraint was only placed on the second output, TESTQ2. The second line of the red status cell indicates the type of the clamp. TESTQ2 is clamped due to the "+incr" constraint. TESTQ1 and TESTQ3 are both vector clamped. The theory of a vector clamp is that the set of recommendations that are made to correct a problem is a vector. The purpose of a vector clamp is to maintain the direction of the vector by adjusting all the recommendations by the percentage that the most restricted output was reduced by. The original recommendation was (31.4, 50, -5.3) and the second output has a maximum positive constraint of 25. Therefore, the TESTQ2 recommendation must be reduced from 50 to 25, which is a reduction of 50%. In order to maintain the direction of the recommendation vector, and to move the process in the right direction, all recommendations are reduced by 50% to (15.7, 25, -2.65). Clicking on the red status field provides a description of the clamp rationale:



Vector clamping has three different modes of operation: implement, advise, and disable. Implement was demonstrated above. Advise mode is demonstrated below (test 12b). The same problem and recommendations were made and the setpoint spreadsheet below along with the warning are presented to the operator. Once the information popup is dismissed, it cannot be recalled.



Finally, disable mode still implements the individual output constraints but there is no vector clamping. The same recommendations and constraints yield the following using (test 12c):

Setpoint Spreadsheet

Wait | Download | No Download | Recalc | Details | Diagnosis

	Action	Outputs	Tag	Current Setpoint	Rec	Final Setpoint	Status	Download Status
1	Active	TESTAPP1 Application						
2	GO	TESTQ1	DiagnosticToolkit/...	9.6	31.4	41		
3	GO	TESTQ2	DiagnosticToolkit/...	23.5	25	48.5	CLAMPED (+ Incr)	
4	GO	TESTQ3	DiagnosticToolkit/...	46.3	-5.3	41		

The next example (test12e) demonstrates what happens when all the recommendations exceed the output constraints. The example below uses the same calculations as examples shown above but with incremental constraints of +/- 2.0 on each output. The original recommendation was (31.4, 50, -5.3), each of which exceeds the incremental change constraint. The changes presented to the operator are shown below.

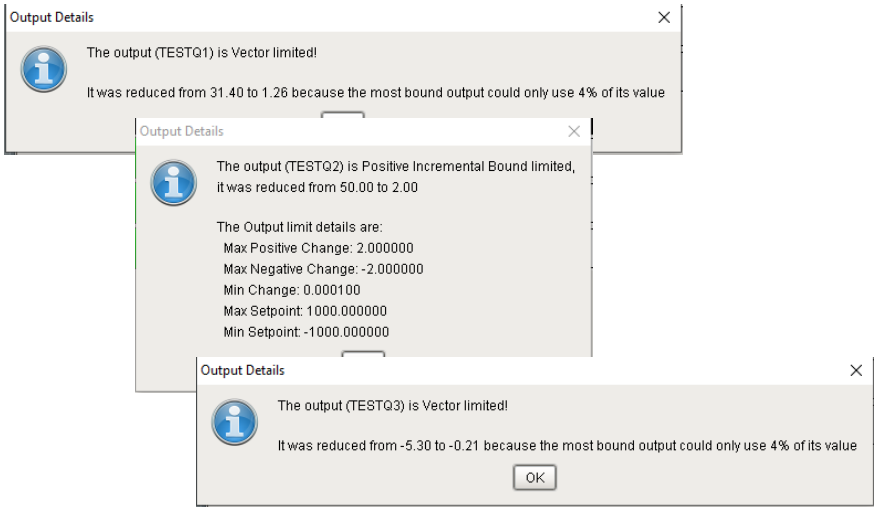
Setpoint Spreadsheet

Wait | Download | No Download | Recalc | Details | Diagnosis

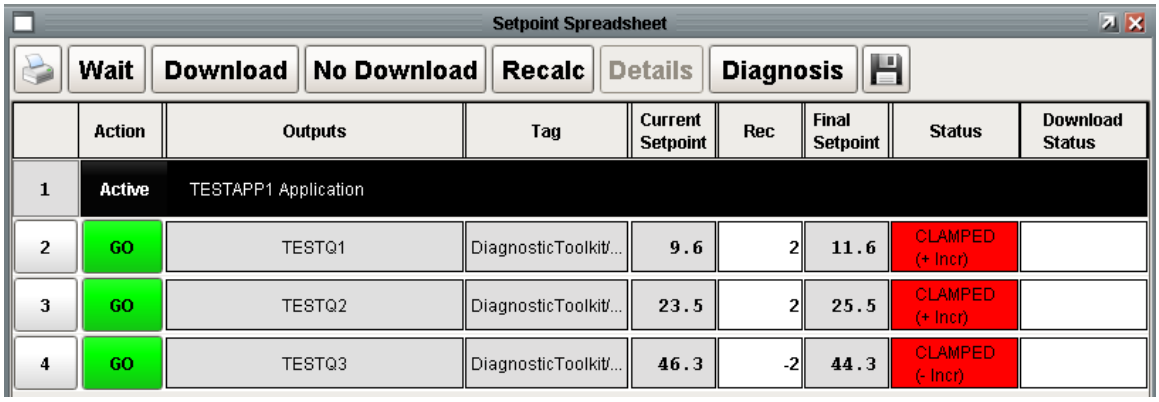
	Action	Outputs	Tag	Current Setpoint	Rec	Final Setpoint	Status	Download Status
1	Active	TESTAPP1 Application						
2	GO	TESTQ1	DiagnosticToolkit/...	9.6	1.26	10.86	CLAMPED (Vector)	
3	GO	TESTQ2	DiagnosticToolkit/...	23.5	2	25.5	CLAMPED (+ Incr)	
4	GO	TESTQ3	DiagnosticToolkit/...	46.3	-0.21	46.09	CLAMPED (Vector)	

When multiple recommendations exceed the output constraints, the diagnostic toolkit determines the most constrained output and then applies vector clamping to the other outputs. The recommendation for TESTQ2 is the most constrained: 50/2 > 31.4/2 >

5.3/2. Therefore, TestQ1 and TestQ3 are vector clamped. The explanations for the clamp rationale are shown below:

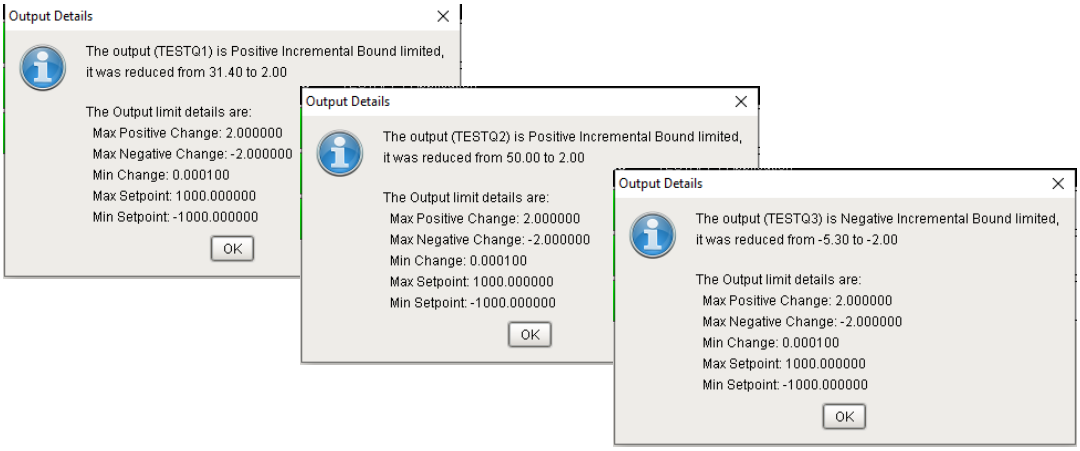


The next example (test12f) demonstrates what happens with the same constraints and exact same recommendations when vector clamping is disabled. The original recommendation was (31.4, 50, -5.3), each of which exceeds the incremental change constraint. The changes presented to the operator are shown below.



	Action	Outputs	Tag	Current Setpoint	Rec	Final Setpoint	Status	Download Status
1	Active	TESTAPP1 Application						
2	GO	TESTQ1	DiagnosticToolkit/...	9.6	2	11.6	CLAMPED (+ Incr)	
3	GO	TESTQ2	DiagnosticToolkit/...	23.5	2	25.5	CLAMPED (+ Incr)	
4	GO	TESTQ3	DiagnosticToolkit/...	46.3	-2	44.3	CLAMPED (- Incr)	

The explanations for the clamp rationale are shown below:



6 Advanced Topics

6.1 Final Diagnosis Processing

The culmination of a diagnostic diagram is generally a Final Diagnosis. The Final Diagnosis block has a logical input and a logical output. The Java engine will evaluate the block as normal. The "action" that this block will perform is implemented in the Python module `ils.diagToolkit.finalDiagnosis.evaluate(block, state)`. The state is expected to be True, False or "unknown". The BLT is event driven, therefore this will be called whenever the state of the final diagnosis, which is the same as the input, changes. The engine will not wait for the Python to complete. If the diagnosis is True,

then the call to `postDiagnosisEntry()` kicks off the main recommendation management logic.

The Python is shown below:

```
    def acceptValue(self,port,value,quality,ts):
        newState = str(value).upper()
        if newState == self.state:
            return

        self.state = newState
        print "FinalDiagnosis.acceptValue: ",self.state

        if self.state != "UNKNOWN":
            print "Clearing the watermark"
            system.ils.blt.diagram.clearWatermark(self.parentuuid)

        handler = self.handler

        # On startup, it is possible for a block to get a value before
        # all resources (like the parent application) have been loaded.
        if handler.getApplication(self.parentuuid)==None or handler.getFamily(self.parentuuid)==None:
            print "FinalDiagnosis.acceptValue: Parent application or family not loaded yet, ignoring state change"
            self.state = "UNKNOWN"
            return

        database = handler.getDefaultDatabase(self.parentuuid)
        provider = handler.getDefaultTagProvider(self.parentuuid)

        print "Using database: %s and tag provider: %s " % (database, provider)

        applicationName = handler.getApplication(self.parentuuid).getName()
        familyName = handler.getFamily(self.parentuuid).getName()
        print "Application: %s\nFamily: %s" % (applicationName, familyName)

        finalDiagnosis = handler.getBlock(self.parentuuid, self.uuid)
        finalDiagnosisName = finalDiagnosis.getName()
        print "Final Diagnosis: %s" % (finalDiagnosisName)

        if self.state == "TRUE":
            print "The diagnosis just became TRUE"
            # Notify inhibit blocks to temporarily halt updates to SQC
            # handler.sendTimestampedSignal(self.parentuuid, "inhibit", "", "",time)
            from ils.diagToolkit.finalDiagnosis import postDiagnosisEntry
            postDiagnosisEntry(applicationName, familyName, finalDiagnosisName, self.uuid, self.parentuuid, database, provider)
        else:
            print "The diagnosis just became FALSE"
            from ils.diagToolkit.finalDiagnosis import clearDiagnosisEntry
            clearDiagnosisEntry(applicationName, familyName, finalDiagnosisName, database, provider)

        # Pass the input through to the output
        self.postValue('out',value,quality,ts)

        print "FinalDiagnosis.acceptValue: COMPLETE"
```

6.2 Data Consistency

In certain situations, there is a concern that related data are consistent. For example, if the raw value for a unit parameter comes from lab data, then at a specific moment in time it may be important that the value from lab data and the filtered value of the unit parameter are consistent. This is relevant when using event based processing of the values in multiple threads. Although the need specifically arose related to lab data and unit parameters, there is a general need to determine if two tags that are related are consistent.

Therefore, a function has been provided as part of the diagnostics toolkit, but can may be used by any toolkit. The tag arguments to the function can be any type of tag. The argument prototype is:

> *checkConsistency(tag1, tag2, tolerance=5, recheckInterval=1.0, timeout=10)* – returns True if abs(the datetime of tag2 - the datetime of tag1) < tolerance, returns False otherwise. All 3 parameters are in seconds.

6.3 Download Processing

The recommendation download is initiated in a client from the setpoint spreadsheet when the operator presses the Download button. The client sends the contents of the setpoint spreadsheet to the gateway via a message. Once the message is received by the gateway, the download proceeds without any further communication to the client. In the gateway, the processing creates a thread for each tag and processes them in parallel. The outcome of the download is communicated back to the client by updating the database. This design provides robustness so that the download will proceed even if the client is disconnected while the download is in process. It also allows multiple clients to observe the download.

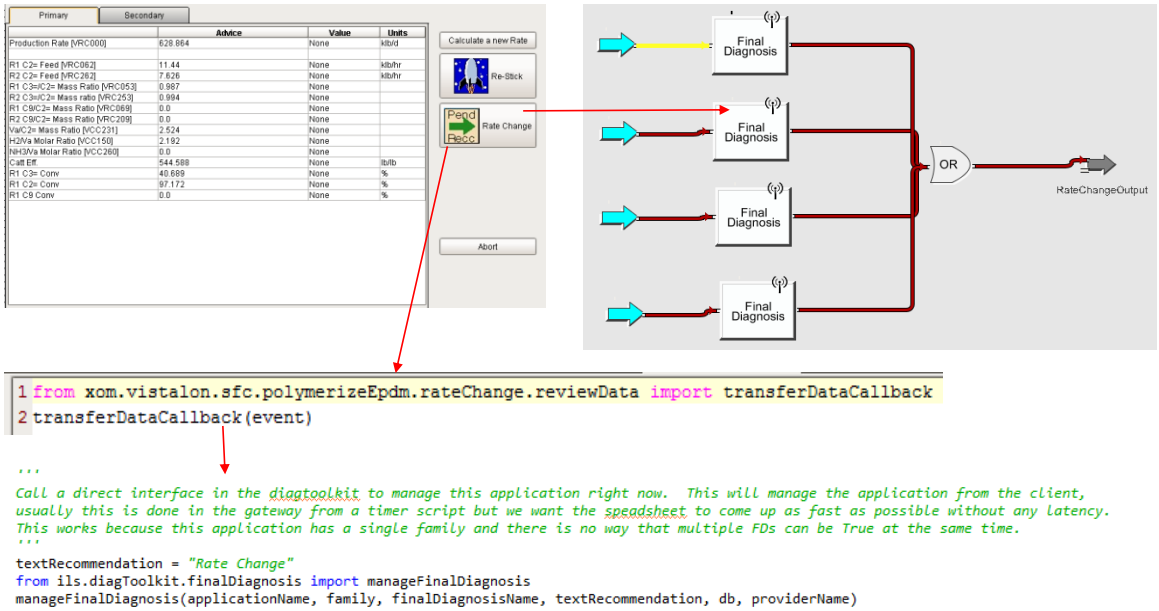
6.4 Triggering a Final Diagnosis Externally

One of the main reasons to trigger a final diagnosis externally is to take advantage of the robust output constraints, user interface, I/O facilities, and logbook and queue messages.

6.4.1 Triggering a Final Diagnosis from a Window

It is common to trigger a Final Diagnosis from a window. The reason for this is to take advantage of the user interface and setpoint safety checks that are built in to the diagnostic toolkit. The process shown below begins with the Review Data window. When the operator presses the "Rate Change" button. The button directly invokes the Final Diagnosis by calling the *manageFinalDiagnosis()* API. This immediately executes the final diagnosis manager without waiting for the final diagnosis to age. It is important to recognize that there is also an input to the Final Diagnosis and an output which goes to an OR block and ultimately to another diagram. For that reason it is necessary to also set the tag input to get the propagation of the value to downstream blocks even though the final diagnosis is being managed manually. If the tag is set without calling

manageFinalDiagnosis() then the final diagnosis will still be managed but there will be a ten second delay while the final diagnosis ages.



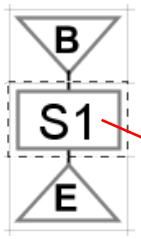

```
1 from xom.vistalon.sfc.polymerizeEpdm.rateChange.reviewData import transferDataCallback
2 transferDataCallback(event)
```

```
'''
Call a direct interface in the diagtoolkit to manage this application right now.  This will manage the application from the client,
usually this is done in the gateway from a timer script but we want the speadsheet to come up as fast as possible without any latency.
This works because this application has a single family and there is no way that multiple FDs can be True at the same time.
'''
textRecommendation = "Rate Change"
from ils.diagToolkit.finalDiagnosis import manageFinalDiagnosis
manageFinalDiagnosis(applicationName, family, finalDiagnosisName, textRecommendation, db, providerName)
```

6.4.2 Triggering a Final Diagnosis from an SFC

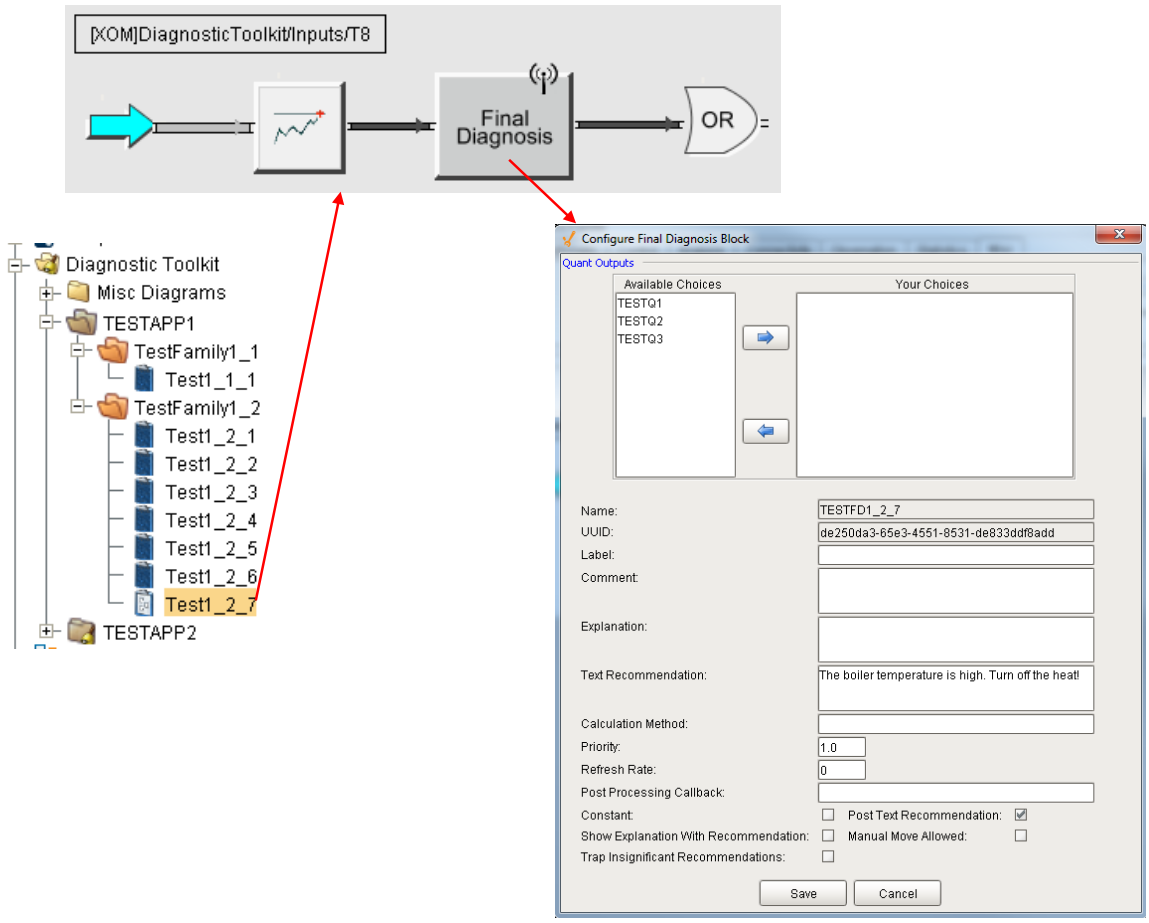
It is common to trigger a Final Diagnosis from a SFC. The process shown below begins with the Action step. The Python calls manageFinalDiagnosisGlobally() which immediately manages the final diagnosis without waiting for the final diagnosis to age. It is important to recognize that there is also an input to the Final Diagnosis and an output which goes to an OR block and ultimately to another diagram. For that reason it is

necessary to also set the tag input to get the propagation of the value to downstream blocks even though the final diagnosis is being managed manually.




```
def onStart(chart, step):
    """
    This will run when the step starts, before any
    other action.

    Arguments:
        chart: A reference to the chart's scope.
        step: A reference to this step's scope.
    """
    print "Trigger a FD"

    from ils.diagToolkit.finalDiagnosis import manageFinalDiagnosisGlobally
    manageFinalDiagnosisGlobally(
        projectName="XOM",
        applicationName="TESTAPP1",
        familyName="TestFamily1_2",
        finalDiagnosisName="TESTFD1_2_7",
        textRecommendation="",
        database="XOM",
        provider="XOM")
```

The final diagnosis that this triggers is shown below. This final diagnosis issues a standard static text recommendation although it could have called a calculation method that calculated a set of new setpoints.



7 Trouble Shooting

Symbolic Ai is an advanced software application with many aspects to it. When trouble shooting the system it is important to distinguish between diagram related problems and recommendation processing related problems. Diagram related problems are everything leading up to a Final Diagnosis step becoming true. Recommendation related problems are everything that happens after a Final Diagnosis becomes true.

7.1 Diagram Related Problems

Call Chuck

7.2 Recommendation Related Problems

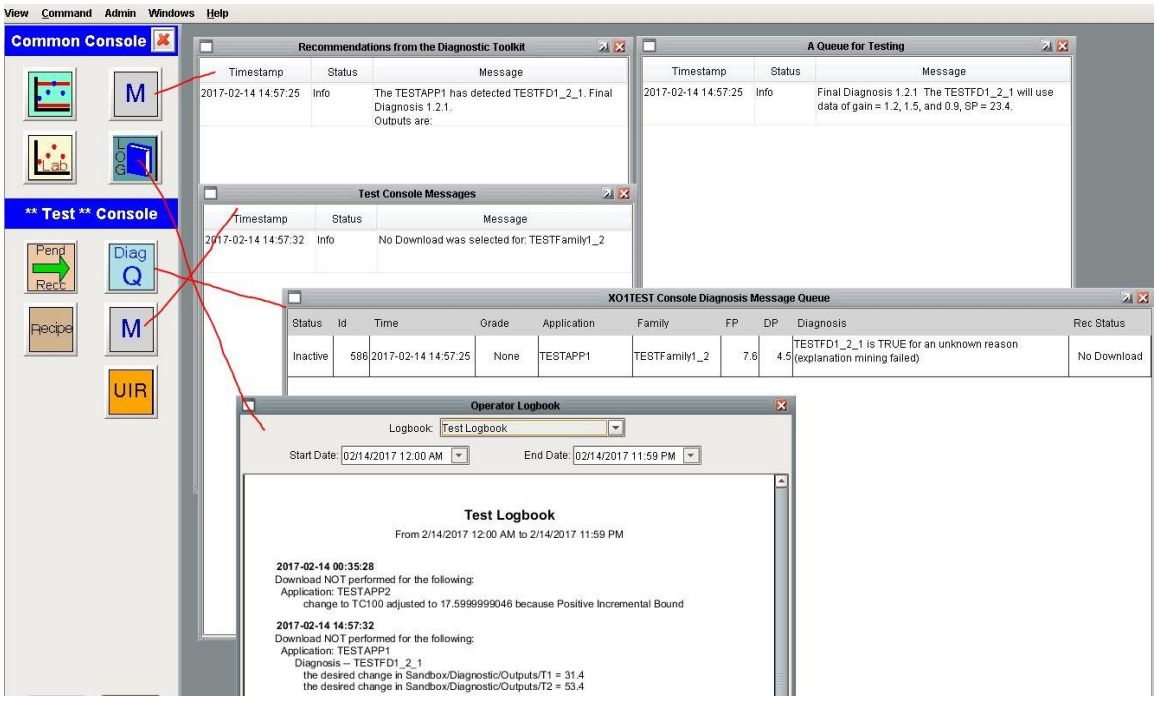
Call Pete

7.3 Logbook and Queue Messages

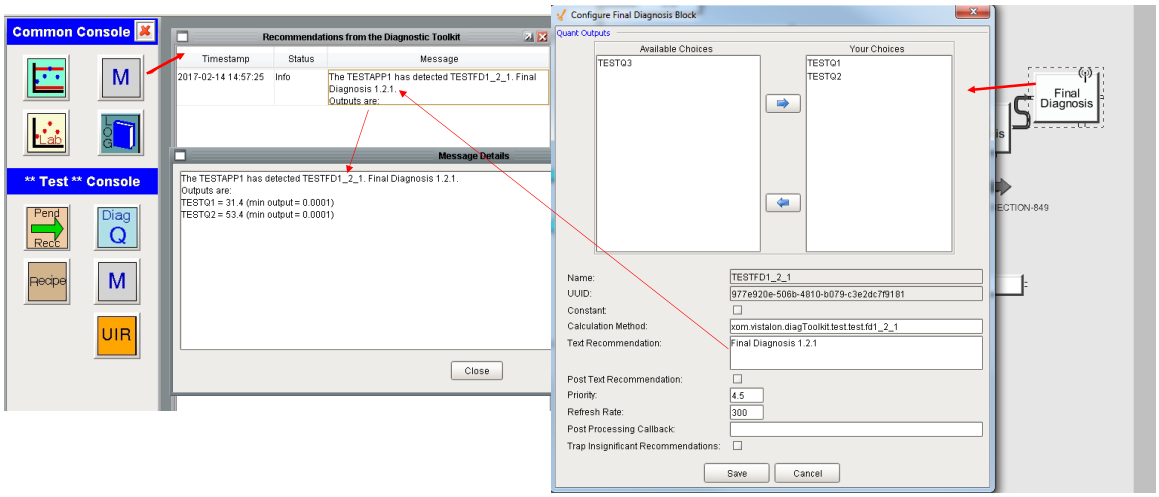
There are numerous facilities set up for logging the actions of the diagnostic toolkit and the actions taken by the operator. It is important to understand what information goes where.

The following diagram demonstrates numerous sources of information that is important for troubleshooting a Final Diagnosis, its recommendations, and the actions around it.

- Recommendation Queue – The top right "M" button on the common console opens a view of the recommendation queue.
- Logbook – The "Log" message on the common console opens a daily view of the Operator Logbook.
- Diagnosis Queue – The "Diag Q" button on the console open the Diagnosis Queue view for diagnosis that are appropriate for the post.
- Console Queue – The bottom right "M" button displays the messages for the console.
- Application Queue – The "Application Queue" view is shown in the top right of the figure. It is accessed from the "View -> Queues" menu



The Recommendation Queue messages warrant a deeper look. The message is constructed from the configuration of the Final Diagnosis.

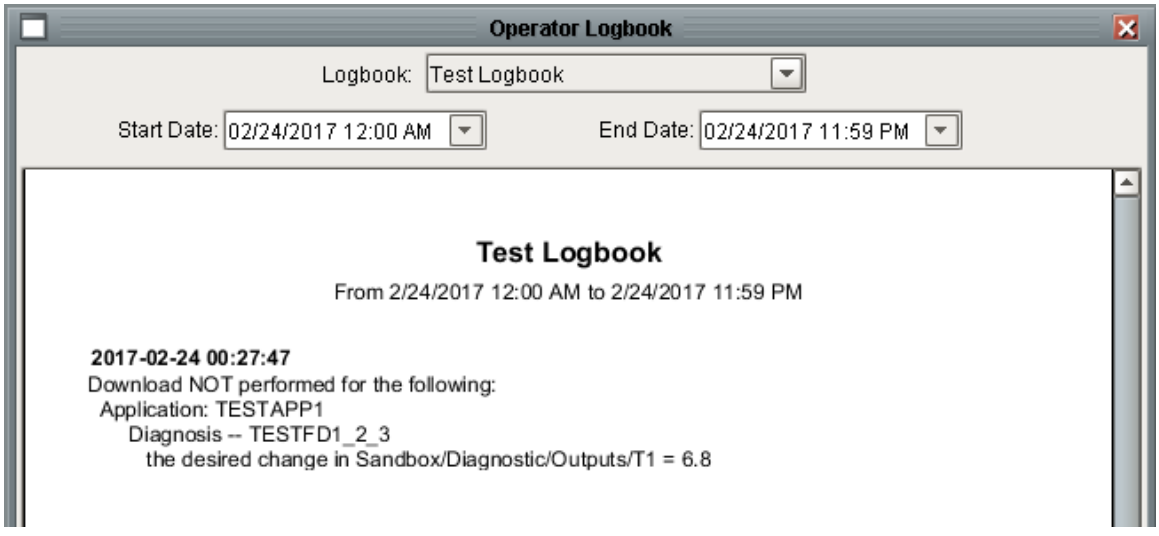


7.3.1 Logbook Message Details

Certain critical actions, such as Download and No Download, are logged to the Operator Logbook. Some features of the logbook are:

- Only the highest priority problems are logged.
- If there is not a recommendation for a quant output then it should not be logged.
- If multiple problems are active, then the individual contributions should be logged along with the aggregate.

The sample shown below is for a problem that had three Quant Outputs but only made a recommendation for one of them.

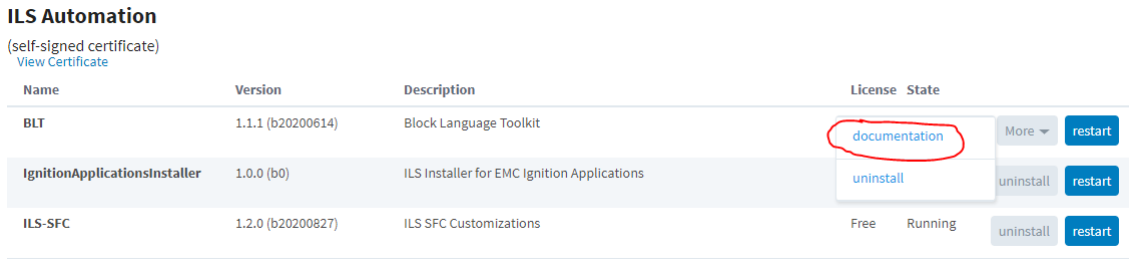


8 Database Tables

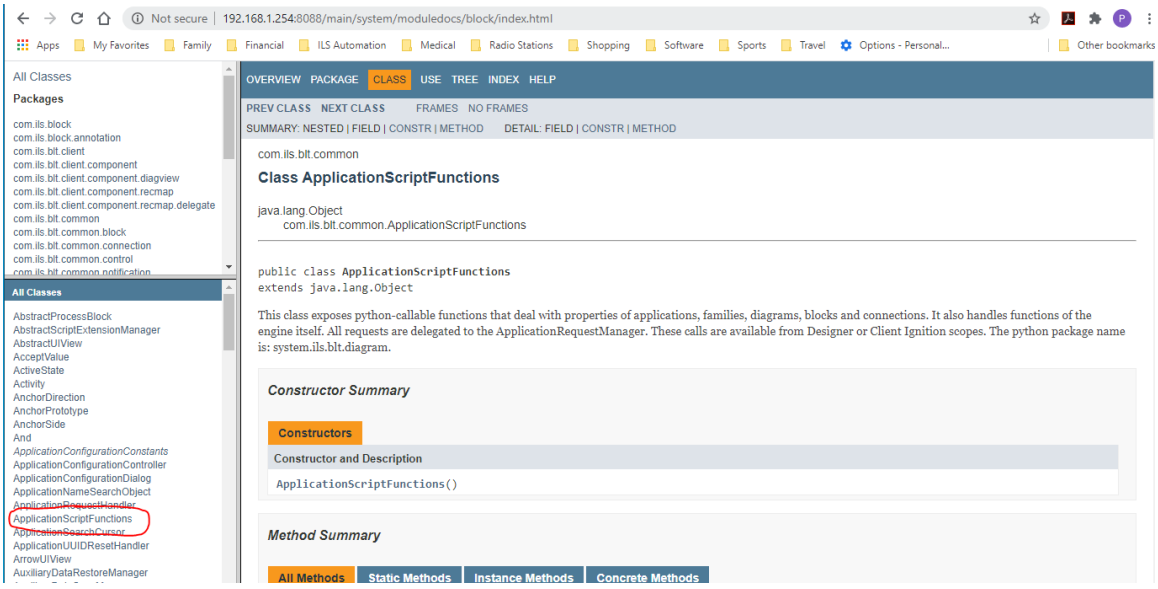
Refer to the EMC Database Design specification for information regarding database tables required by Symbolic Ai.

9 Programmer's Interface

When writing calculation methods or other Python scripts, it is often necessary to interact with blocks in the diagnostic schematic. There are a number of interfaces for programmatically collecting, resetting, and interrogating blocks. The programming interface is completely described in Javadocs which can be accessed from the gateway webpage. Accessing it is a little difficult. From the Configure tab, select the Modules page, scroll down to the BLT module under ILS Automation, press *More* and select *documentation*.



From the main Javadocs page, select the class ApplicationScriptFunctions under the "All Classes" section.



In order to use any of the scripting functions, you need to first import:

import system.ils.blt.diagram as diagram