

---

# IPautomata

---

Designer

---

This document contains material that is the proprietary property of IPSoft. Disclosure outside IPSoft is prohibited except by licensed or other confidentiality agreement.

Copyright © 2013 IPSoft Inc. All rights reserved.

IPSOFT PROPRIETARY

## Contents

1.	HIGH LEVEL ARCHITECTURE .....	1
2.	MESSAGE FLOW .....	1
3.	TERMINOLOGY .....	1
3.1	Categories.....	1
3.2	Live.....	1
3.3	Archived .....	2
3.4	Templates/Automata Classes .....	2
3.5	(Ticket) Matchers.....	2
3.6	Execution Mode .....	2
3.7	Connection .....	2
3.8	Variable.....	2
3.9	State.....	3
3.10	Normal State.....	3
3.11	Phone Call State.....	3
3.12	Wait State.....	3
3.13	Link State .....	3
3.14	Fork State.....	3
3.15	Join State.....	3
3.16	Success State.....	4
3.17	Transition.....	4
3.18	Execution.....	4
3.19	Execution Status.....	4
3.20	Execution Pointer .....	4
3.21	Execution Pointer Status .....	4
3.22	Approval Status.....	5
4.	IPAUTOMATA DESIGNER .....	5
4.1	Automata View.....	6
4.2	Properties View .....	7
4.2.1	Core .....	7
4.2.2	Connections .....	8
4.2.3	Matchers .....	9
4.2.4	Variables.....	10
4.2.5	Approval View .....	12
4.3	Shells View.....	13
4.4	Connecting.....	13
4.5	Designer View.....	16
4.5.1	Recording.....	17
4.5.2	Manual Construction.....	17
4.5.3	Editing States.....	17
4.6	Testers.....	53
5.	WEB GUI .....	56
5.1	Radar Widget.....	56
5.2	Automaton Details .....	58
5.3	Execution Search .....	59
5.4	Execution Details.....	60
5.5	Approvals.....	61

## Document History

Author	Version	Date	Comments
Dlosey	1.0	July 12, 2012	Twiki version
Communications	1.1	July 26, 2013	Formatted, minor edits
Communcations	1.2	July 29, 2013	A few grammar/spell tweaks

## 1. High Level Architecture

IPautomata is made of three main components:

- IPautomata Designer: Automata are built using the designer. The designer is a Java Swing client launched over Java Web Start. It communicates back to IPcenter over Spring's http(s) based remoting protocol. Use of the designer is restricted to IPSOFT users with the "ipautomata\_admin\_user" IPcenter role. The designer requires connectivity (over 443) to IPcenter.
- IPautomata Executioner: A standalone (Java) daemon that can execute multiple automata concurrently. The executioner connects over ssh and/or lsh to remote hosts through a chain of proxy hosts.
- IPautomata web gui: A HTML GUI where automata can be executed, past executions examined, etc. Automata cannot be changed through the web GUI.

## 2. Message Flow

1. A new incoming alert is persisted by the IPradar daemon and the ticket ID is published on the ipcenter.ipradar.newticket JMS topic.
2. The IPautomata Executioner picks up this ticket ID and locates matching automata. An execution and initial execution pointer are created for any matching automata whose execution mode is automatic. For each created execution a message containing the execution pointer ID is sent to the pcenter.ipautomata.signal.pointer JMS queue.
3. The IPautomata Executioner picks up the execution pointer ID from the ipcenter.ipautomata.signal.pointer queue and starts executing.
4. At any time a user can create an execution through the web GUI. The web GUI simply creates the execution and sends the pointer ID to the executioner as if it had been created in step 2.

## 3. Terminology

### 3.1 Categories

Categories are simply a way to organize automata. They have no other effect.

### 3.2 Live

An automaton can either be Live or Inactive/not live. Only Live automata can be executed.

### 3.3 Archived

An automaton becomes archived when 1) it has been executed one or more times and 2) a change is made to it through the designer. This is done so examination of past executions always reflect the state of the automaton at that time.

### 3.4 Templates/Automata Classes

An automaton with its template flag set is simply an automaton that can be used as a template. It cannot be executed. There is significantly less validation done when saving a template (variable values are not required, etc.)

### 3.5 (Ticket) Matchers

An automaton has one or more ticket matchers. A ticket matcher consists of a field from the ticket and a regular expression to apply against that field's value. If more than one matcher is configured for a given automaton, the match type can be specified as either AND or OR. If AND, then all matchers must match for the automaton to match the ticket. If OR, then one or more must match. A CMDB Query can also be used in the "Ticket Field" and the expression can be used to match on the result of that query. For example, putting /OS Type in the Ticket Field, and Linux in the expression, will match when a host is Linux.

### 3.6 Execution Mode

There are currently two execution modes. Manual execution mode requires that an automaton be initiated by an engineer. Automatic execution mode allows an automaton to be executed automatically if it matches and incoming ticket.

### 3.7 Connection

A connection is the set of information required to connect to a target host. This includes all proxy hops, protocols, usernames, etc. are required to reach the target host.

### 3.8 Variable

Variables allow automata to have different behavior at runtime based on the conditions at the time of execution. Variables can be of several types.

Regardless of type a variable can be flagged as secure. A secure variable will be encrypted in the database and never displayed in the web GUI or execution logs. They can be used for interactive passwords or other sensitive information. Every variable can also have a notes field which can be used to describe the purpose of the variable.

See the [variables section](#) below for details on each type of variable.

## 3.9 State

A state consists of a single action (currently) and 0 or more incoming and outgoing transitions. A state without any incoming transitions is considered the root of an automaton. There can only be a single root in the automaton.

### 3.10 Normal State

A normal state consists of single action to be executed. The action will be executed immediately by the executioner and the appropriate outgoing transition followed. Examples are executing a host command, sending an email, updating a ticket.

### 3.11 Phone Call State

A phone call state consists of a single point in a telephone conversation. Upon entry into its Phone Call Action, a call will be placed and a message spoken, or a message will be spoken on an already established Phone Call. Each action may prompt for touch tone responses, which can be noted and processed in transitions.

### 3.12 Wait State

Upon reaching a wait state the executioner daemon may or may not perform an action, but it does not take an outgoing transition, but rather waits until an external system, thread, whatever, signals the execution to continue. An example is waiting for a user to fill out a form in which case the form processor signals the execution to continue after the form is submitted.

### 3.13 Link State

A link state links to another automaton. Variables can be passed in and/or extracted from the sub-automaton. Links may be of arbitrary depth and number, however, **recursion is not supported** (don't try it). The return code is always 0 from a link state.

### 3.14 Fork State

A fork state splits the execution into two or more concurrent paths of execution. Both paths will execute concurrently until they reach a join state.

### 3.15 Join State

A join state joins two or more execution paths. Upon reaching a join state that path of execution is paused until all incoming transitions have been traversed.

### 3.16 Success State

Each automaton must have one or more Success states. If the last state executed by an automaton is a success state, the entire execution is considered a success. Otherwise, the execution is considered to have failed to resolve the problem.

### 3.17 Transition

A transition has a source and target state and a transition evaluator. A transition evaluator consists of an input item and an expression. If the expression is true, the transition is followed.

### 3.18 Execution

An instance of an automaton execution.

### 3.19 Execution Status

An execution moves through the following statuses:

1. SCHEDULED - The automata is set to automatic and the execution is scheduled to start in the future.
2. READY - The execution is created and waiting for the executioner to pick it up
3. RUNNING - The executioner is currently executing the execution
4. COMPLETE - The executioner has finished executing the execution and completed without error.
5. ABORT\_REQUESTED - A user has requested to abort the execution
6. ABORTED - The executioner has aborted the execution
7. AUTO\_ABORTED - The execution was scheduled, but was automatically aborted due the ticket being resolved, etc.
8. FAILED - The execution failed unexpectedly

### 3.20 Execution Pointer

An execution pointer points to an executing state. There may be several execution pointers for a single execution if the automaton contains forks, links, or wait states.

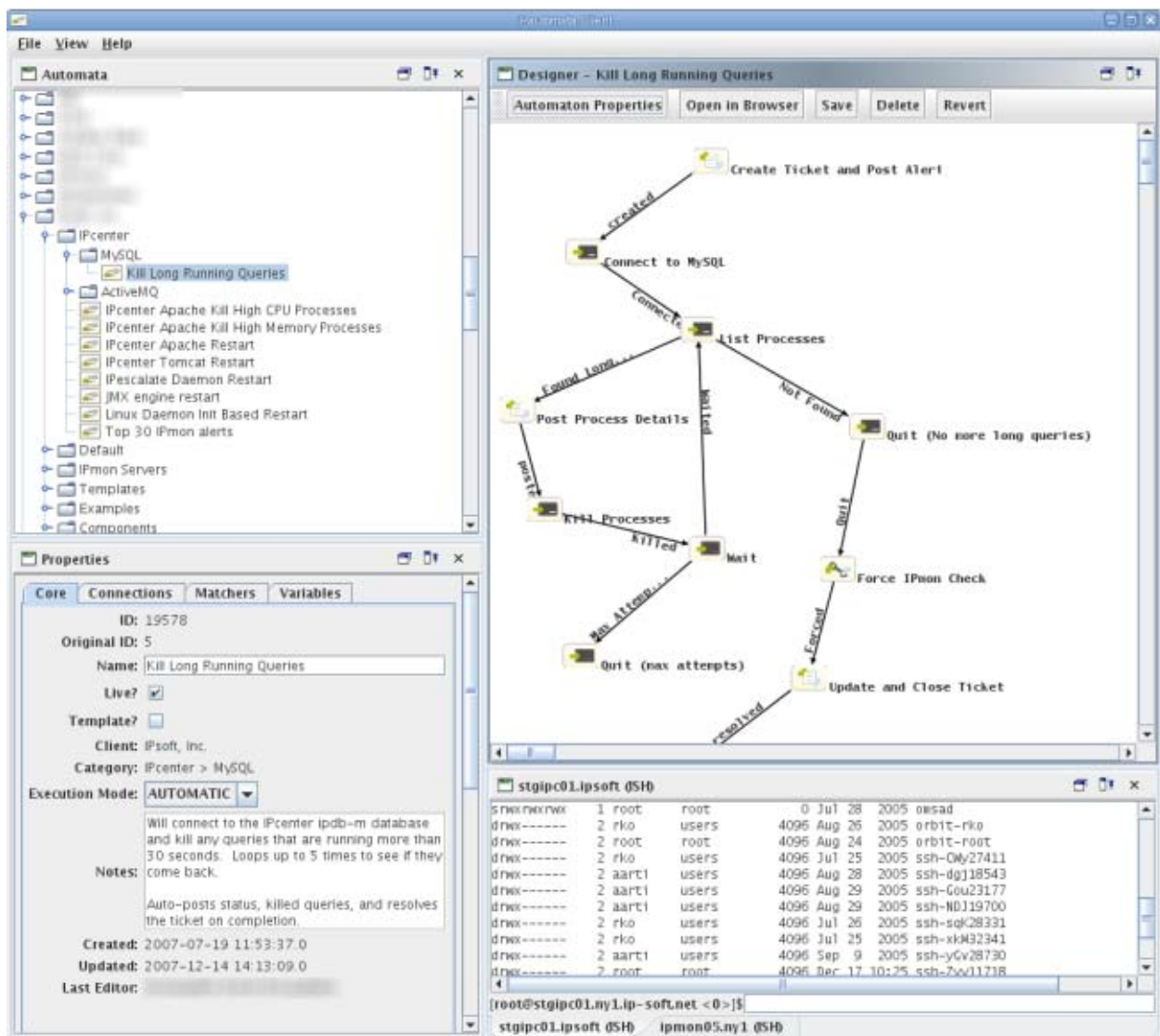
### 3.21 Execution Pointer Status

1. RUNNING - The pointer is currently running
2. COMPLETE - The pointer is complete
3. JOIN\_WAIT - The pointer has reached a join, but not all other required pointers are there yet.
4. FAIL - The pointer has failed. You can recover the pointer by clicking the recover link.
5. LINK\_WAIT - The pointer is waiting for the linked automaton to complete.
6. LINK\_EXIT - Transient internal state used to synchronize the exit from a link.
7. WAIT - The pointer is waiting for an external event (Form Action, etc.)

## 3.22 Approval Status

1. NONE - assigned when an automaton is created
2. PENDING - this automaton is now waiting to be reviewed by a member of the [CSI](#) team. If it is a non-escalation automaton, it can only be executed by the creator of the automaton or a user w/ the authority to approve the automaton.
3. APPROVED - this automaton has been approved and can be executed by anyone with `AUTHORITY_IPAUTOMATA_EXECUTE`.
4. REJECTED - this automaton has been rejected

## 4. IPautomata Designer



The main screen of the designer is split into four dockable frames.

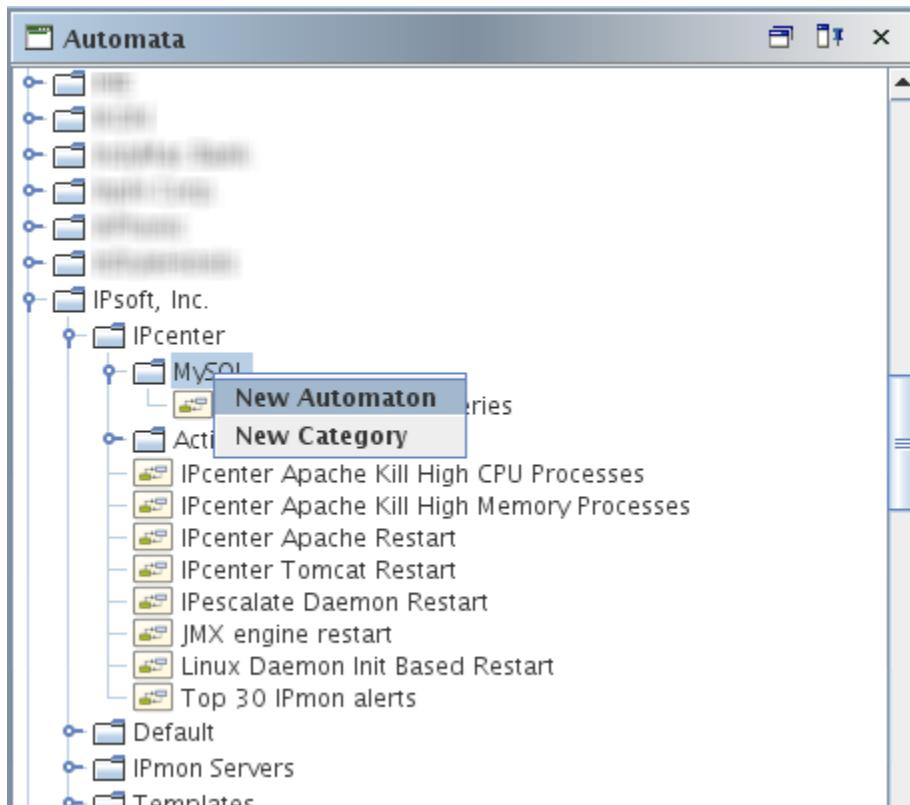
IPSOFT PROPRIETARY



- Automata - A tree view of all available automata organized by client and category.
- Designer - Shows the graph of the currently open automaton.
- Properties - Selection sensitive properties and global properties of the open automaton.
- Open Shells - Tabbed frame of open remote shells.

Note that dockable frames can be moved, docked, maximized, etc. to your liking. The "Automata" and "Properties" frames can be closed and reopened via the "View" menu.

## 4.1 Automata View



Automata are organized by client and category. To create a new automaton or sub-category, right click on a category.

An automaton can be moved between categories (under the same client) by dragging.

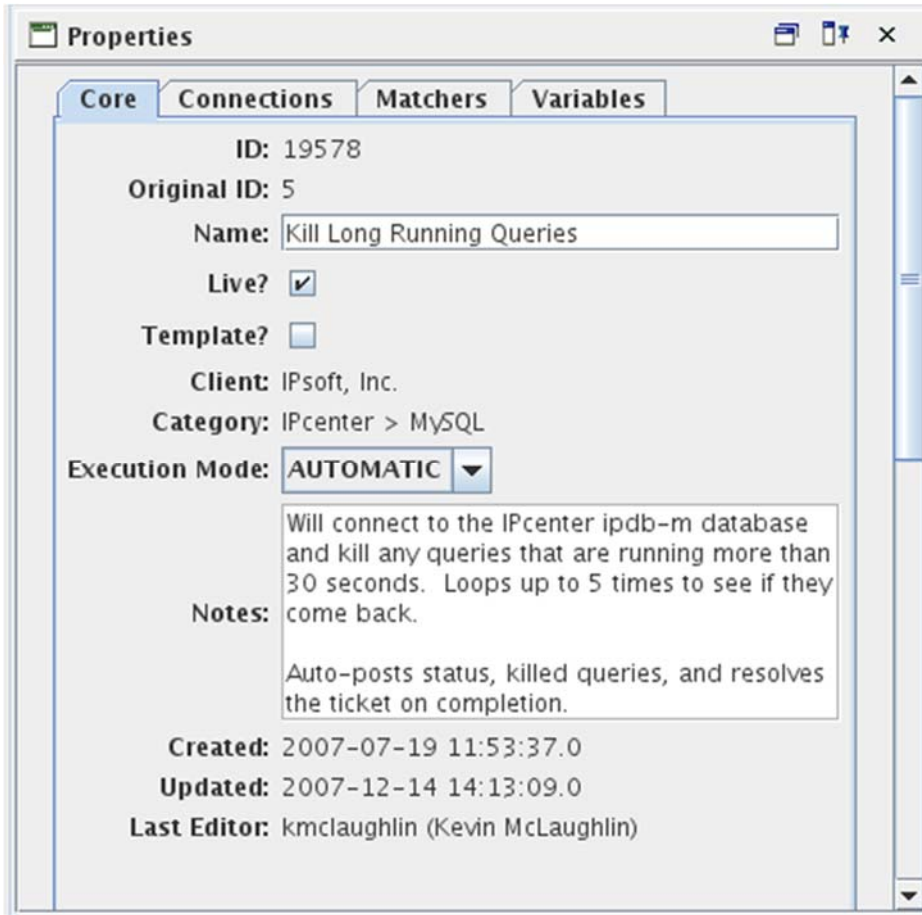
An automaton can be copied, even across clients, by holding down ctrl and dragging to the new parent category.

Double clicking an automaton will open the staged version if present, and the approved one if not. By right clicking an automaton name you can select to open either the staged or approved version if present. When an approved automaton is saved, it is automatically copied into a staged version where edits can be made

and tested. Once the staged version is submitted for approval and is approved, it will become live. Any edits to a staged automaton will not be automatically executed until it is approved.

## 4.2 Properties View

### 4.2.1 Core



**Properties**

**Core** | Connections | Matchers | Variables

ID: 19578  
Original ID: 5

Name: Kill Long Running Queries

Live? ☒  
Template? ☐

Client: IPsoft, Inc.  
Category: IPcenter > MySQL

Execution Mode: AUTOMATIC ▼

Notes:  
Will connect to the IPcenter ipdb-m database and kill any queries that are running more than 30 seconds. Loops up to 5 times to see if they come back.  
Auto-posts status, killed queries, and resolves the ticket on completion.

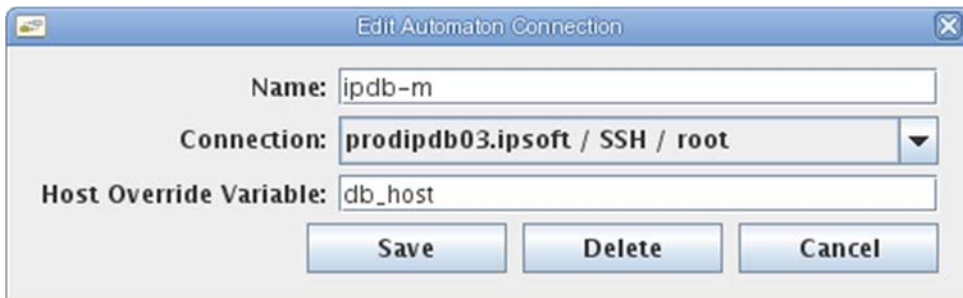
Created: 2007-07-19 11:53:37.0  
Updated: 2007-12-14 14:13:09.0  
Last Editor: kmclaughlin (Kevin McLaughlin)

This view contains the core properties of the open automaton. You can always return to this view by clicking the 'Automaton Properties' button at the top of the designer view.

## 4.2.2 Connections

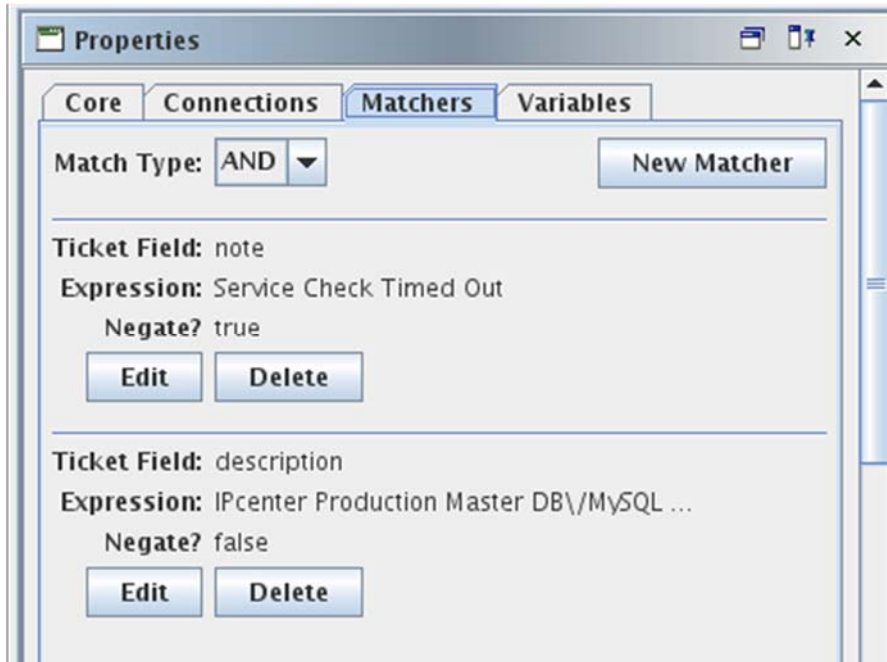


Add/Edit connections used by this automaton. Note that you do not edit the actual connection information here, since that is not specific to the open automaton. Connection information can be edited through the connect dialog.

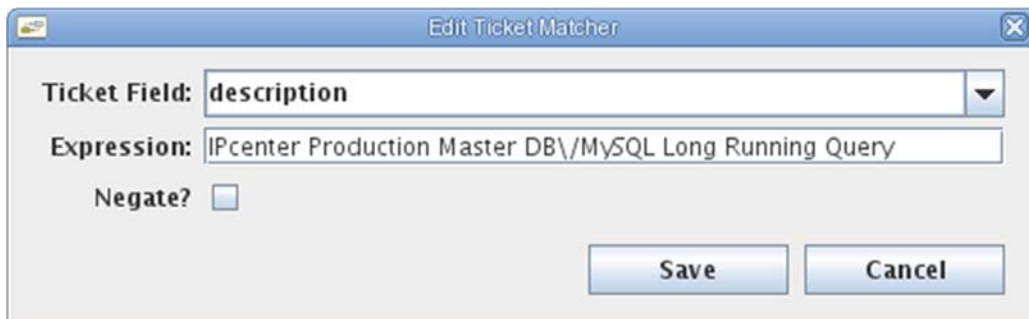


The edit connection dialog is fairly straightforward, with the exception of the "Host Override Variable" field. If this field is specified a variable is created under that name. At runtime the name/alias of the target host will be overridden with the value of the variable. This allows a single automaton to manage a group of identical hosts. For example, to manage a cluster of web servers, you would likely extract the actual host from the alert.

### 4.2.3 Matchers



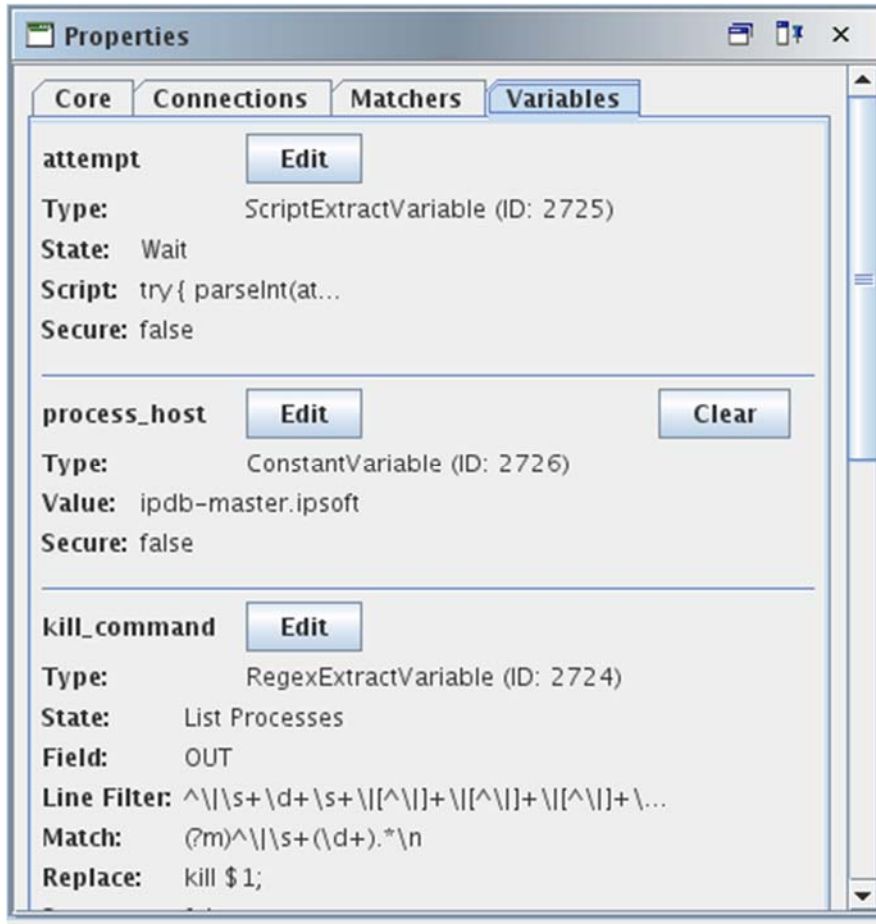
Add/Edit (ticket) matchers for the open automaton. If match type is AND then all matchers must match the ticket. If OR, then one more matchers must match for the automaton to be match.



The Edit Ticket Matcher dialog is fairly straightforward. Select the field from the ticket to match against and specify the regular expression. Matching is done via [Matcher.find\(\)](#). See [Pattern documentation](#) for more information about the valid regular expression format.

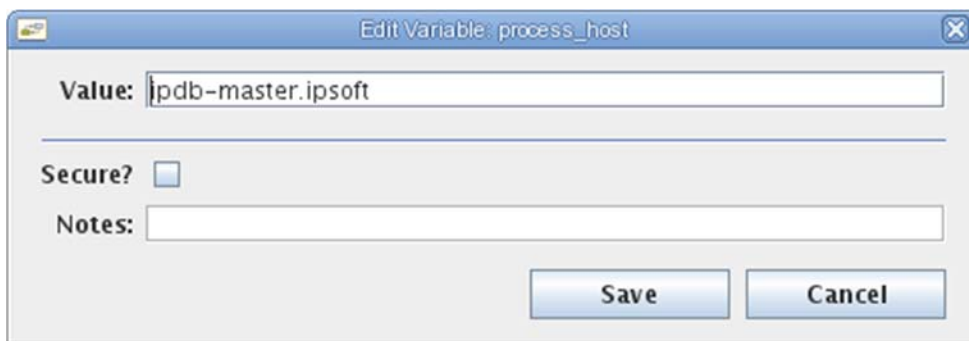
A predefined set of ticket fields is given in the drop down. You are not restricted to this list, however. If you wish to access a field that is not listed, ask for the appropriate field name. For example, "attributes(foo)" allows access to the IPmon attribute named "foo". Matching using CMDB query is also possible, like /Monitored Address in Ticket Field.

#### 4.2.4 Variables



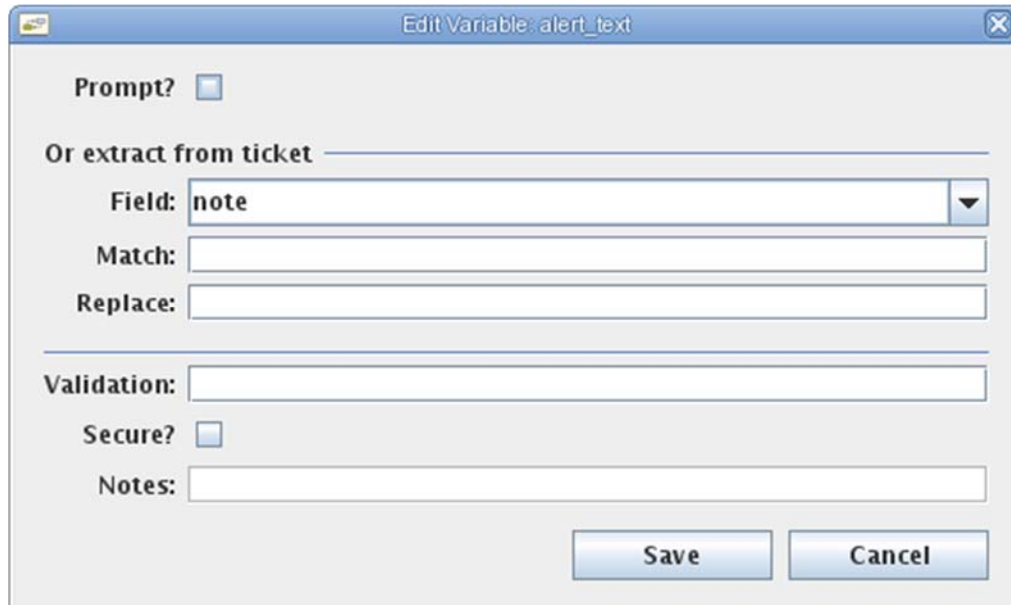
All variables defined in the open automaton are shown in this view. Variables without a value will have "Constant" and a "Runtime" buttons to define the type and value of the variable. To change the type of a variable, click the "Clear" button. If the "Clear" button is not available, the variable is extracted from a state's output and must be edited through the state's properties.

#### Constant Variables



There are only two values for a constant variable. The value and whether the variable is secure. Secure variable's values will never be displayed in the web GUI or logged. Variables for passwords, etc. should be marked as secure.

## Runtime Variables



Runtime variables come in two flavors.

If "Prompt?" is checked, the value of the variable will be gathered from the user at execution time.

If prompt is not selected the value of the variable will be extracted from the selected ticket field. If the "Match" field is empty, the value of the variable will equal the extracted field. If, however, "Match" is specified and is a valid regular expression it will be used along with the "Replace" pattern to manipulate the value of the ticket field.

The field is manipulated as follows if a "Match" expression is given:

```
<code>
```

```
Pattern p = Pattern.compile(matchPattern);
```

```
return p.matcher(fieldValue).replaceAll(replacePattern);
```

```
</code>
```

[Pattern documentation](#)

A predefined set of ticket fields is given in the drop down. You are not restricted to this list, however. If you wish to access a field that is not listed, ask for the appropriate field name. For example, "attributes(foo)" allows access to the IPmon attribute named "foo".

Finally, a runtime variable can also be marked secure to prevent logging of its value and display in the web GUI.

## State Extract Variables

See [Editing States](#) for a description of editing extract variables.

### 4.2.5 Approval View



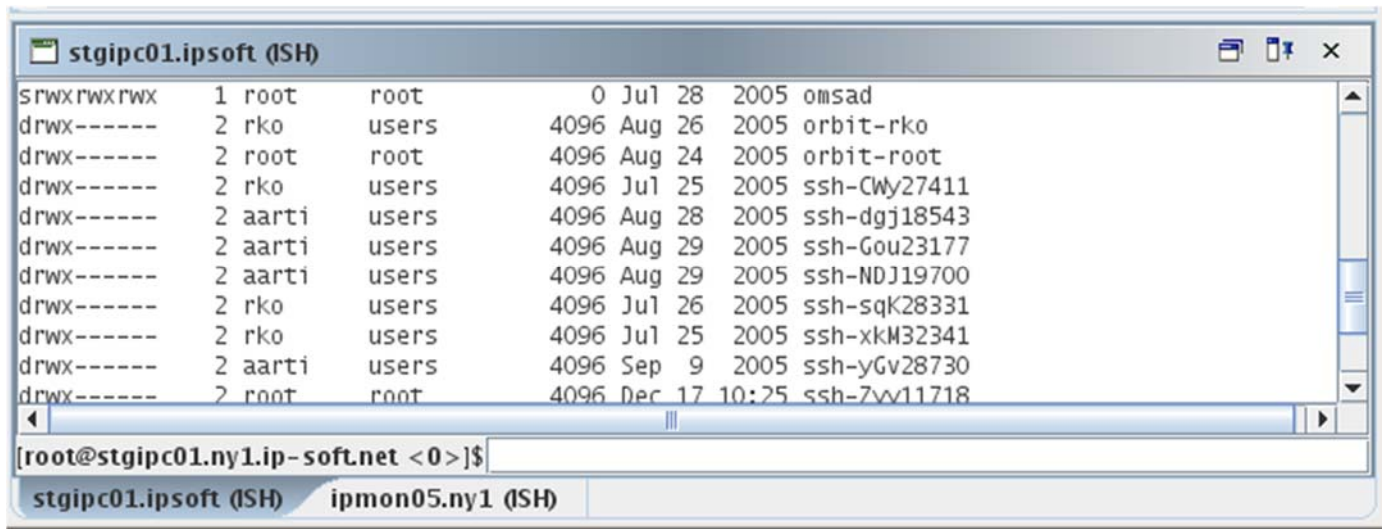
The screenshot shows the 'Automaton Properties' dialog box with the 'Approval' tab selected. The status is 'PENDING'. The 'Notes' field contains 'Please approve'. The 'Submitted' timestamp is '2009-10-26 12:49:45.0' and the 'Submitted by' is 'kmclaughlin (Kevin McLaughlin)'. The 'Reviewed' and 'Reviewed by' fields are empty. At the bottom, there are three buttons: 'Submit', 'Approve', and 'Reject'.

The approval view shows the current (as of opening) approval status of the automaton. From this view, you can:

- Submit the automata for approval. This should be done once your automata has been tested and is ready to become executable by others.
- Approve or Reject the automata. These button will be enabled for people with approval authority for new automata.

The notes field should be used to deliver any comments/explanation/etc. between the submitter and the approver of the automaton.

## 4.3 Shells View



The shells view contains your open shells. The name of the frame is the host (alias) you are connected to followed by the protocol. The command line is at the bottom and contains your effective user, the host (as returned by hostname) and the return code of the last executed command. The command line supports most "readline" shortcuts and has its own basic internal history. There is no tab completion.

When connected over lsh, do **not** run any interactive commands such as top, vi, tail -f, etc.

When connected over ssh, interactive commands are OK, however, realize that there is no terminal emulation so programs like vi will be nearly impossible to use. Also, realize that although it is an interactive shell, it is still basically a batch shell. That means commands that send continuous output and never terminate **must** be avoided or your shell will never return. If you happen to run a command that does not return, you must login to the remote box and kill your command manually. Such is life. At least for now...

## 4.4 Connecting

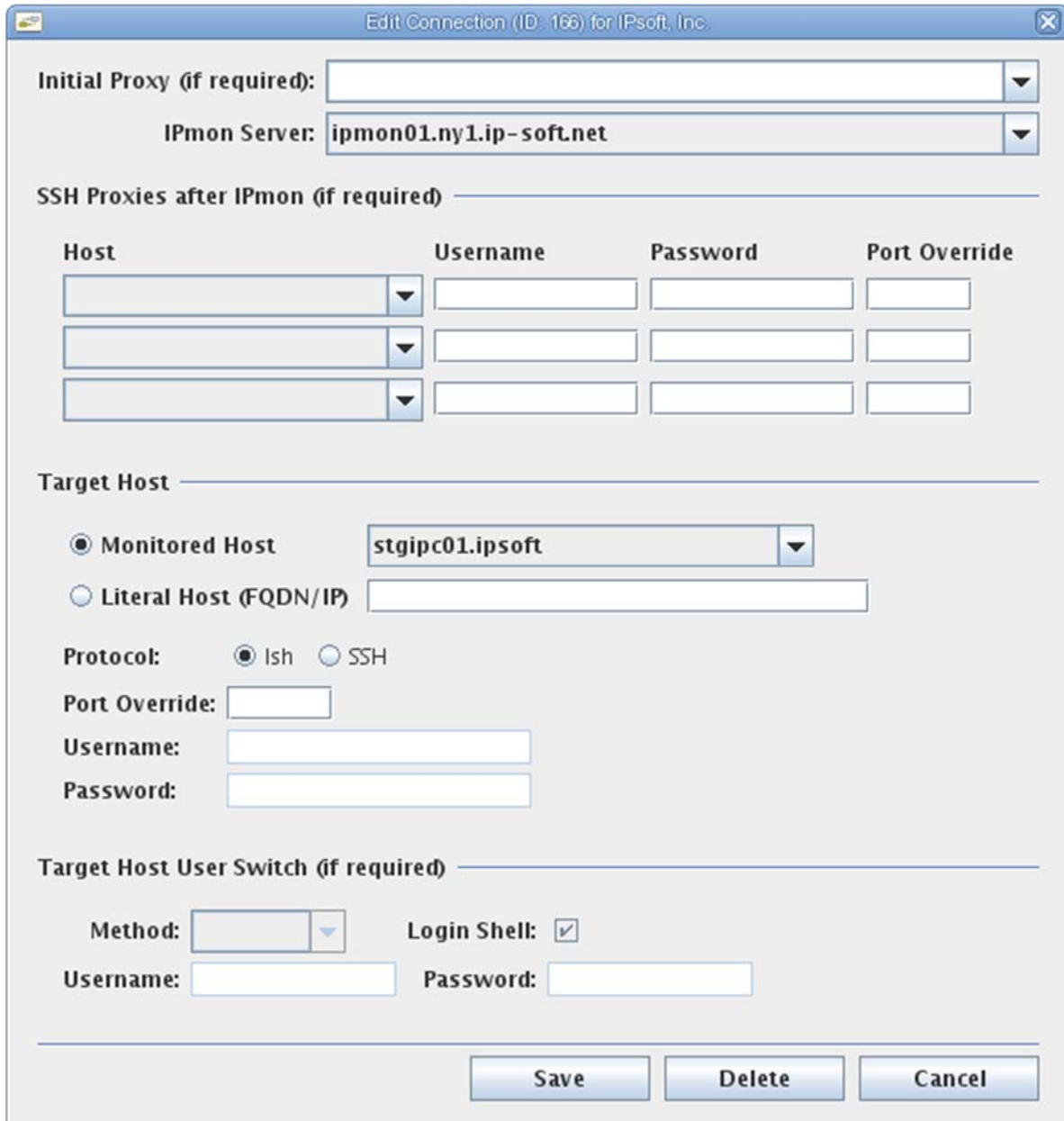
To begin connecting to a remote host, choose File->Connect.





Upon choosing a client from the drop down the "Connection" drop down will populate with all available connections for the selected client. If a required connection is not there, click New (and see below). Otherwise, enter your login credentials to IPmon and click connect. A status window will display the progress as the connection is established. Depending on how many hops there are between you and the target host, it may take a few seconds to establish the connection.

To edit a connection, select the connection and click "Edit. Then, see below, **especially** for the caveats.



Dialog box: Edit Connection (ID: 166) for IPSOFT, Inc.

Initial Proxy (if required):

IPmon Server:

SSH Proxies after IPmon (if required)

Host	Username	Password	Port Override
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Target Host

☒ Monitored Host

☐ Literal Host (FQDN/IP)

Protocol: ☒ Ish ☐ SSH

Port Override:

Username:

Password:

Target Host User Switch (if required)

Method:

Login Shell: ☒

Username:

Password:

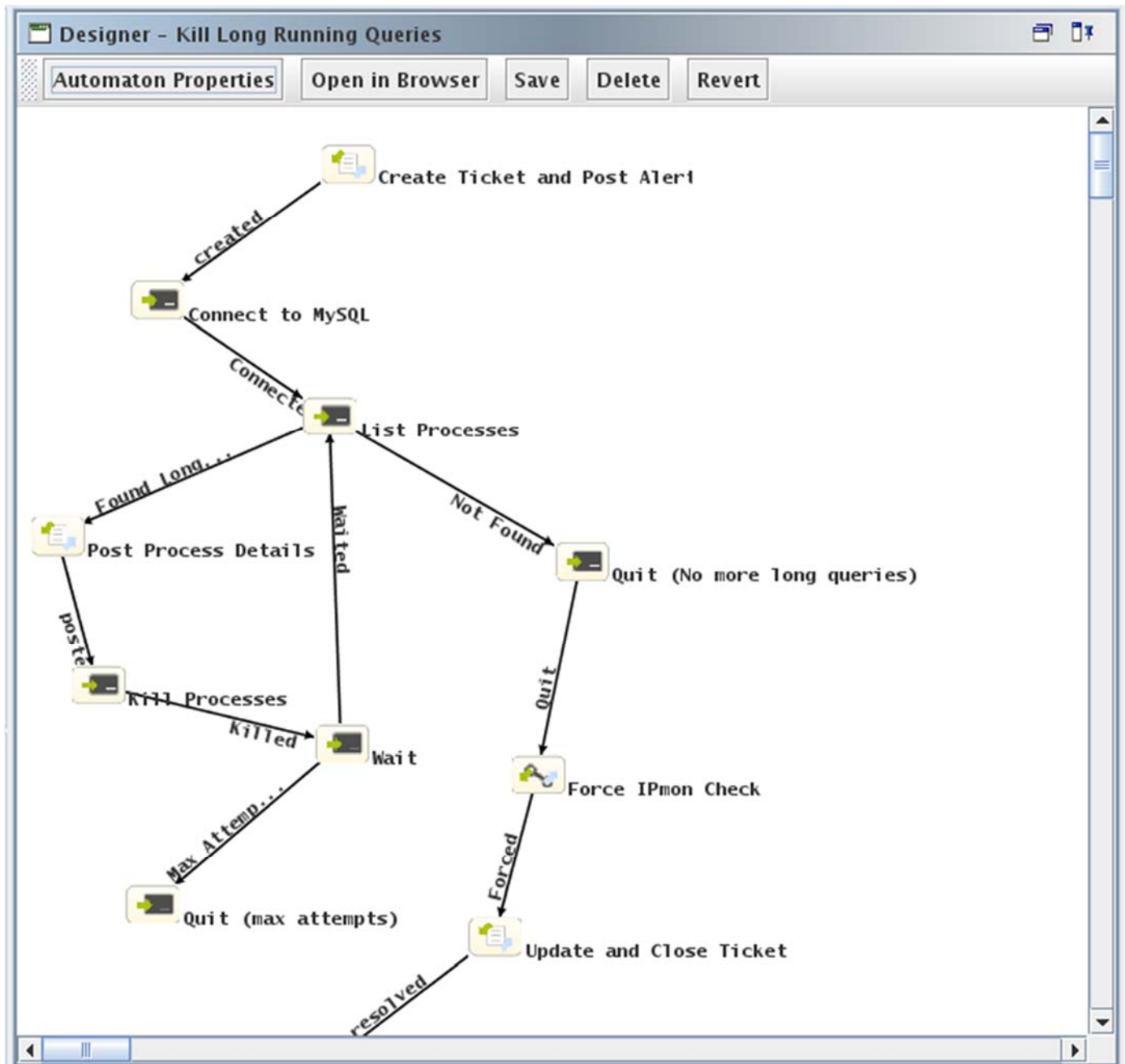
Buttons: Save, Delete, Cancel

When editing a connection, especially one that may be in use by existing automata, do not change the effective user of the final shell on the target host. Create a new connection to the same host instead.

The edit connection dialog may look complex, but most attributes are not required unless the hops are numerous.

- Initial Proxy (if required): If the desired IPmon Server is not directly reachable, select (or enter) another host that can reach it. In most cases this is not required.
- IPmon Server: Select the IPmon server that is monitoring the host you want to reach. The value of this field influences the rest of the "Host" fields in the dialog.
- SSH Proxies after IPmon: If the target host is not directly reachable from the IPmon server, up to 3 more hosts can be tunneled through over SSH. Select the host and populate the user and password required to login to each.
- Target Host: Select the target host from the monitored host drop down (where you want a shell) or enter the FQDN if the host is not monitored. Choose the protocol. Prefer lsh over SSH unless you need to run interactive commands, or commands as a user other than the user running IPremote. Note if IPremote is running as root, you may still be able to run your commands via `su - user -c 'blah'`. lsh is faster and much simpler than SSH. If you must use SSH, select SSH and fill in the username and password of the user to login as.
- Target Host User Switch: If using SSH, you can switch (once) to another user directly after logging into the target host. Choose the method and populate the required credentials and whether it should be a login shell ( - ).

## 4.5 Designer View



The designer view presents a graphical view of the graph that makes up an automaton. Both states and transitions should be named appropriately for a state's function and a transition's condition. For example, "Kill Apache" is better than "kill -9 \${pid}" for a state that kills apache. And "Killed" is better than "RET =~ 0" for the transition when the "kill" is successful.

### 4.5.1 Recording

Recording builds up the graph by watching commands executed in one or more open shells. To begin recording, select an existing state and right click and select "Start Recording". If no states exist, right click anywhere in the designer view. To stop recording, right click in an empty space in the designer view and selected stop recording. Recording can be stopped and started as many times as required. When starting recording from an earlier state, however, the first transition must be manually completed since the return code of the start state no longer exists in the shell. It is also possible to pause and resume recording. When paused, the return code of the last recorded command is preserved so that upon resuming the transition is automatically created.

### 4.5.2 Manual Construction

In addition to recording an automaton, one must add some state types manually by right clicking and selecting the "New State" menu followed by the action type. All action types except [HostCommandActions](#) must be added manually.

To add a transition manually, select two states and right click on one of them. Choose the appropriate transition (direction) that you would like to add. If the right click is not working, hover your mouse on the source state and wait few seconds. Once the mouse pointer changes to hand shape, click the right mouse button and drag it to target action. You will get the transition. Edit it accordingly.

To delete a state, right click on it and select delete. All incident transitions will also be deleted. Note that if you delete an internal state you must construct transitions manually so that the automaton only has a single root.

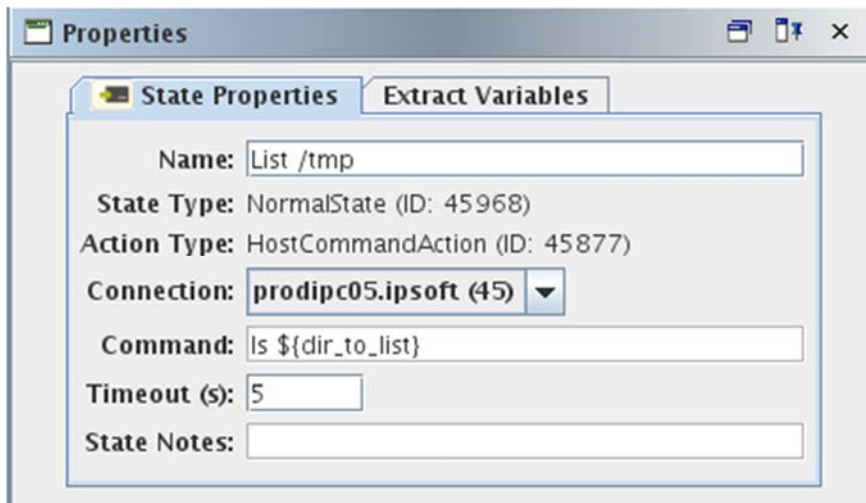
### 4.5.3 Editing States

To edit a state's properties, select a single state. The state's properties will be visible in the Properties View. The properties available vary based on the state and action type.

All states have the following attributes:

- Name - Give the state a meaningful name. "Kill Apache" vs. "kill -9 1234".
- State Type - The type of state followed by the state's ID.
- Action Type - The type of action associated with state followed by the action's ID.
- State Notes - A description of the purpose of the state.

#### Normal State - Host Command Action



A Host Command Action executes a single command.

For recorded states, the name will default to the command. Change it, if desired, to something more meaningful than a cryptic shell sequence.

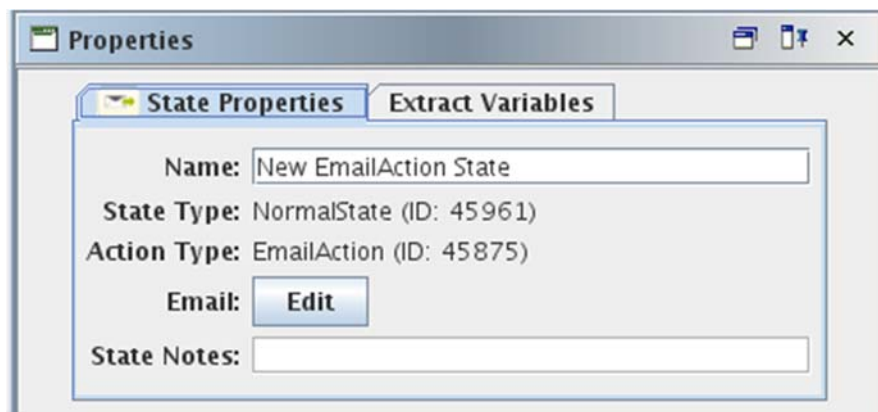
Connection specifies the connection (from the Connections tab in automaton properties) that this state will use.

Timeout is a very crude timeout implemented only for commands that go over ssh connections at this time. It should be left at the default unless it is expected that the command will take longer **without** generating any output. For example, a long SQL query could be considered to have its timeout increased.

The following return codes are possible:

- Any return codes from the host itself
- -999 : Unexpectedly unable to retrieve a return code from the host

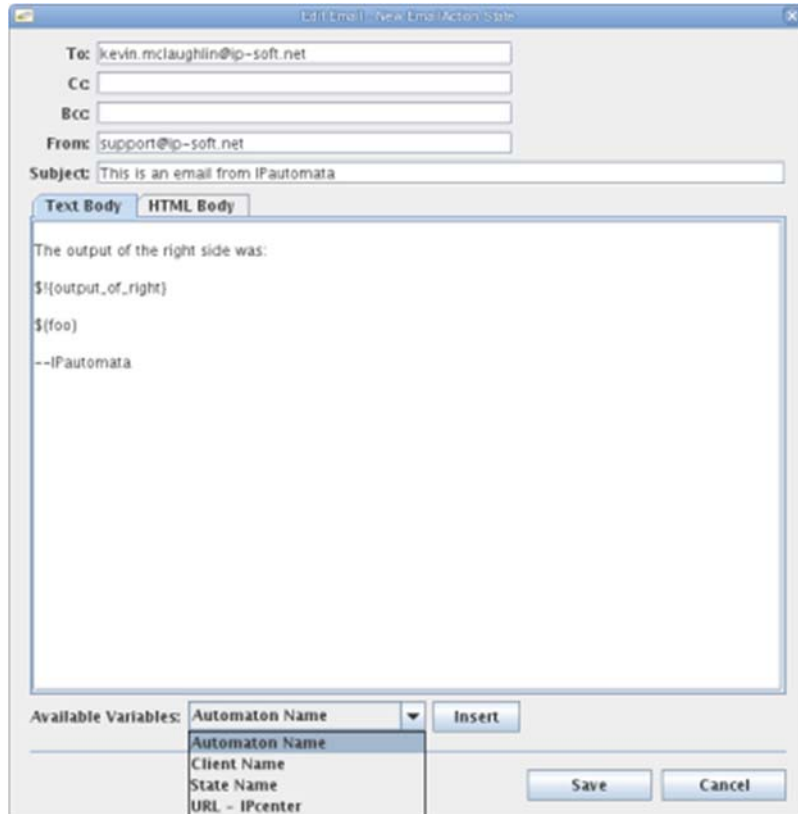
### Normal State - Email Action



An email action allows sending an email.

The action returns 0 if the email was sent successfully or 1 if there was an error.

Click "Edit" to fill out the email fields.



Email Fields:

- To: A ", " delimited list of email recipients
- Cc: A ", " delimited list of email recipients
- Bcc: A ", " delimited list of email recipients
- From: The from address for the email
- Subject: The subject of the email

You can use variables in any of the above fields via the `${var_name}` syntax.

Fill in the text body of the email and optionally the HTML body. If no HTML body is specified the email will be sent as plain text only.

Use the "Available Variables" drop down to insert a variable into the body of the email. If you want to define a variable within the email (rather than using an existing variable) simply type it in with the `${var_name}` syntax and the variable can be populated via the Variables tab within the automaton's properties.

## Normal State - Update Ticket Action

Properties

State Properties

Extract Variables

Name:

Create Ticket and Post Alert

State Type:

NormalState (ID: 45793)

Action Type:

UpdateTicketAction (ID: 45747)

Ticket Attributes

IPpm Post:

Edit

Subj/Desc:

ETA (min):

Owner

Select

Var

None

☒ No Change

Department

Select

Var

None

☒ No Change

Client

Select

Var

None

☒ No Change

Radar Queue

Select

Var

None

☒ No Change

Radar Status

Select

Var

None

☒ No Change

Tx Notes:

State Notes:



The Update Ticket Action allows an automaton to create or update an **IPpm** ticket and change various fields of the IPradar ticket.

The following return codes are possible:

- 0 : IPpm ticket was created/updated
- 1 : There was an error updating the IPpm ticket
- 2 : There was no radar ticket associated with the execution, so no IPpm ticket was created/updated

**When using the update ticket action, be cautious with what changes you make to the IPradar ticket.** This actions allows changing things (status, queue, etc.) that can affect the workflow built into IPradar. For example, you could change the owner to None, put a ticket pending client, and set the ETA out 6 months. Clearly that would cause problems.

Click "Edit" to edit the IPpm post (see below). All posts to IPpm will be done as the IPautomata user.

If the subject field is filled out the subject of the ticket will be changed. Variables can be used in the subject via the `${var_name}` syntax.

If the eta field contains an integer, the ETA will be set to that many minutes in the future. If the field contains a variable (`${var_name}`) the ETA will be set to the variable's value minutes in the future. Otherwise, the ETA will not be changed.

The transaction notes field (at the bottom) allows specifying the radar transaction notes for the update.

The rest of the fields have one or more of the following options:

- Select: Choose a constant value from the drop down
- Var: Extract the value from a variable (just use the variable name, not `${var_name}`)
- None: Set the value to 'None'
- No Change: Do not change the value



Post for New IPpm Ticket | Post for Existing IPpm Ticket

Queue: ☒ Select IPsoft-Ops ☐ Var

Post

There are long running queries in the IPcenter database: \${process\_host}

The alert is is:

\${alert\_text}

The queries will be killed. Stand by:

Thanks,

@automata

Available Variables: Automaton Name Insert

Save Cancel

If creating a new ticket, fill in the post content. Variables can be inserted into the post content via the "Available Variables" drop down or manually with the `${var_name}` syntax.

Select the target IPpm queue from the drop down or specify the variable (just the name) to extract the queue from. By using a variable it is possible to use the `default_ippm_queue` attribute from an IPmon alert.

Post for New IPpm Ticket | Post for Existing IPpm Ticket

Queue: ☐ Select  ☒ Var

☐ None ☒ No Change

Post Blank to use new post content

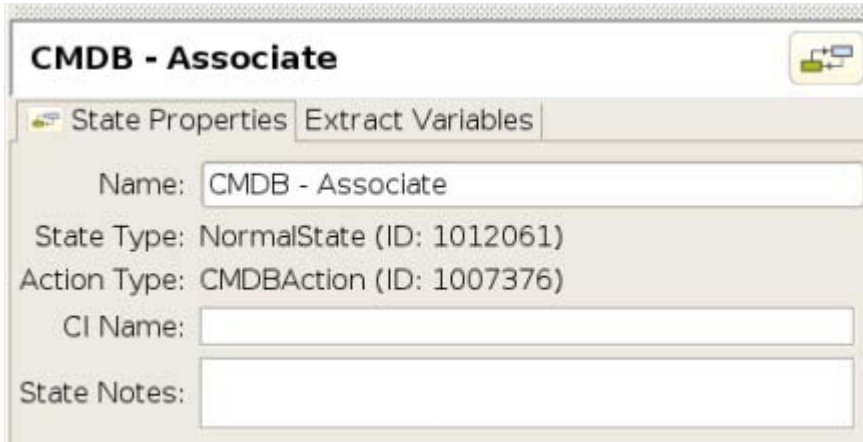
Available Variables: Automaton Name Insert

Save Cancel

If updating a ticket, fill in the post content. If the post content is blank, the content from the new post will be used. Variables can be inserted into the post content via the "Available Variables" drop down or manually with the `${var_name}` syntax.

Select the target IPpm queue. The various options behave the same as for the new post, with the additional option of not changing the queue at all.

### Normal State - IPcmdb Action

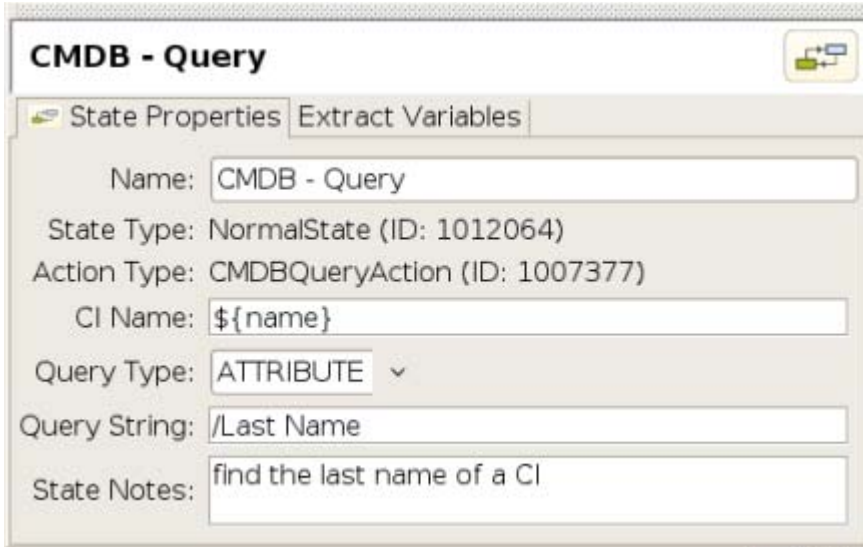


An IPcmdb Action associates a single Configuration Item (CI) from IPcmdb with the Automaton's Radar Ticket. It takes one parameter, the name of the CI to link. Upon success, no response is passed through OUT or ERR, just the successful return code. Non-successful codes will indicate the reason for failure in ERR.

The following return codes are possible:

- 0 : CI was associated (or already had been associated) with the [RadarTicket](#)
- 1 : There was no CI with that name found in the CMDB
- 2 : There was no radar ticket associated with the execution, so no CMDB association can be made
- -1: Unexpected Error

### Normal State - IPcmdb Query



An IPcmdb Query Action allows you to specify a Configuration Item (CI), and then to query for attributes of that CI, or for other CIs related to that CI. The following input is all required:

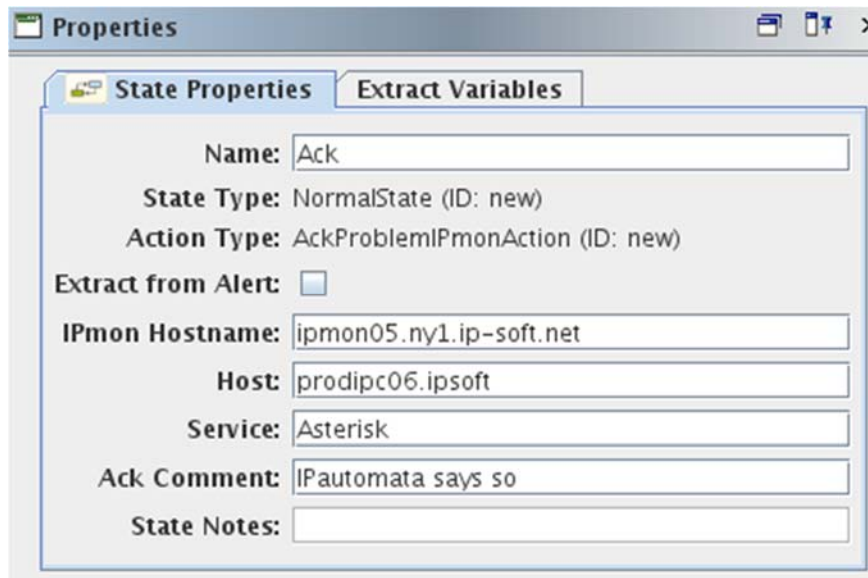
- CI Name: The name of the CI to use as the basis of this search
- Query Type: Must be either ATTRIBUTE or CI.
- Query String: The search query, in [CMDBQueryTool](#) format.

An ATTRIBUTE query will return to OUT all values for that particular attribute (if any are found), one line per value.

A CI query will return to OUT the names of all CIs that match the relationship specified in the query, one line per name.

- 0 : CI and ATTRIBUTE/QUERY ran successfully and found results
- 1 : There was no CI with that name found in the CMDB
- 2 : The base CI is found, but the ATTRIBUTE/CI query found no results
- -1: Unexpected Error

### Normal State - IPmon - Ack Problem



The IPmon Ack Problem action allows an automaton to acknowledge a problem in IPmon.

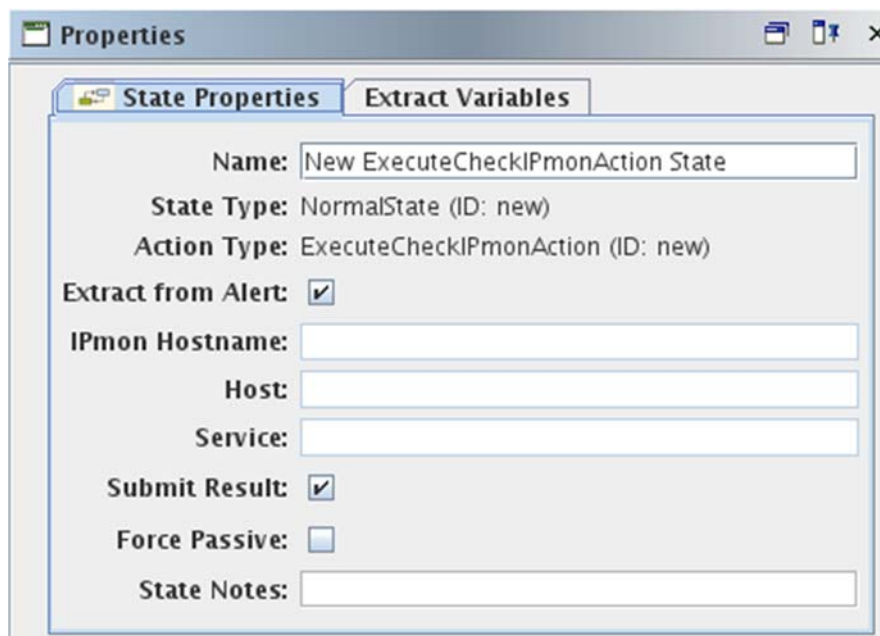
Possible Return Codes:

- 0 - Success
- 20 - Extract from alert set, but no ticket was associated with the execution
- 21 - Extract from alert set, but no IPmon alert associated with the ticket
- Anything else is an unexpected error

If the "Extract from Alert" checkbox is selected, the IPmon Hostname, Host, and Service values will be extracted from the IPmon alert associated with the radar ticket. If not selected, IPmon Hostname and Host are required. Variables may be used with the `${var_name}` syntax.

Enter the reason for the Ack in the "Ack Comment" field. This value will be visible in IPmon.

#### **Normal State - IPmon - Execute Check**



The screenshot shows a 'Properties' dialog box with two tabs: 'State Properties' and 'Extract Variables'. The 'State Properties' tab is selected. It contains the following fields and controls:

- Name:** New ExecuteCheckIPmonAction State
- State Type:** NormalState (ID: new)
- Action Type:** ExecuteCheckIPmonAction (ID: new)
- Extract from Alert:** ☒
- IPmon Hostname:** [Text Field]
- Host:** [Text Field]
- Service:** [Text Field]
- Submit Result:** ☒
- Force Passive:** ☐
- State Notes:** [Text Field]

The IPmon Execute Check action allows an automaton to execute a check configured within IPmon and optionally feed the result of the check into IPmon as a passive result. The check is executed by IPautomata (not by IPmon with schedule immediate).

Possible Return Codes:

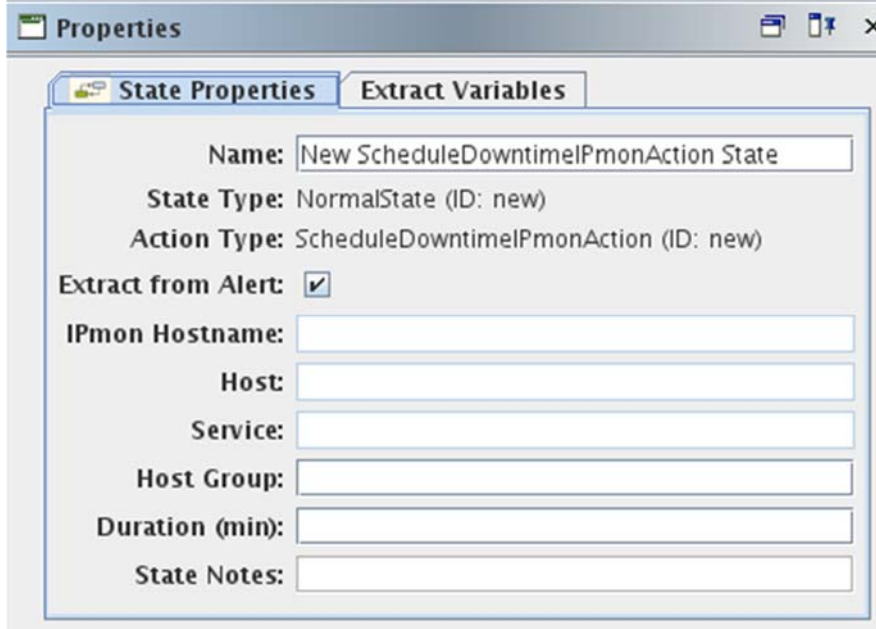
- -1 - IPmon status UNKNOWN
- 0 - IPmon status OK
- 1 - IPmon status WARNING
- 2 - IPmon status CRITICAL
- 11 - If there is an error finding the host, service, or actually executing the check on the IPmon host
- 20 - Extract from alert set, but no ticket was associated with the execution
- 21 - Extract from alert set, but no IPmon alert associated with the ticket
- Anything else is an unexpected error

If the "Extract from Alert" checkbox is selected, the IPmon Hostname, Host, and Service values will be extracted from the IPmon alert associated with the radar ticket. If not selected, all three fields are required. Variables may be used with the `${var_name}` syntax.

If the "Submit Result" checkbox is selected, the IPmon status returned by the check will be fed into IPmon as a passive result. Latency for IPmon to process the result is typically less than 10s.

By default checks that appear to be passive checks (check interval set to 'none') will not be executed by this action. If you know that the check can be executed as it is configured in IPmon, you can select the "Force Passive" checkbox. Do **not** do this unless you know what you are doing and are sure that the check can be executed in this manner.

## Normal State - IPmon - Schedule Downtime



The screenshot shows a 'Properties' dialog box with two tabs: 'State Properties' and 'Extract Variables'. The 'State Properties' tab is active. It contains the following fields and options:

- Name:** New ScheduleDowntimeIPmonAction State
- State Type:** NormalState (ID: new)
- Action Type:** ScheduleDowntimeIPmonAction (ID: new)
- Extract from Alert:** ☒
- IPmon Hostname:** [Empty text box]
- Host:** [Empty text box]
- Service:** [Empty text box]
- Host Group:** [Empty text box]
- Duration (min):** [Empty text box]
- State Notes:** [Empty text box]

The IPmon Schedule Downtime action allows an automaton to schedule downtime for a set of hosts, services, or hostgroups.

Possible Return Codes:

- 0 : OK
- 1 : Incorrect usage
- 20 - Extract from alert set, but no ticket was associated with the execution
- 21 - Extract from alert set, but no IPmon alert associated with the ticket
- 255 : Error in schedule-downtime.pl script
- Anything else is an unexpected error

If the "Extract from Alert" checkbox is selected, the IPmon Hostname, Host, and Service values will be extracted from the IPmon alert associated with the radar ticket. If not selected, all three fields are required. Variables may be used with the `${var_name}` syntax.

The "Host Group" cannot be extracted from the alert. You must supply the value, or use a variable.

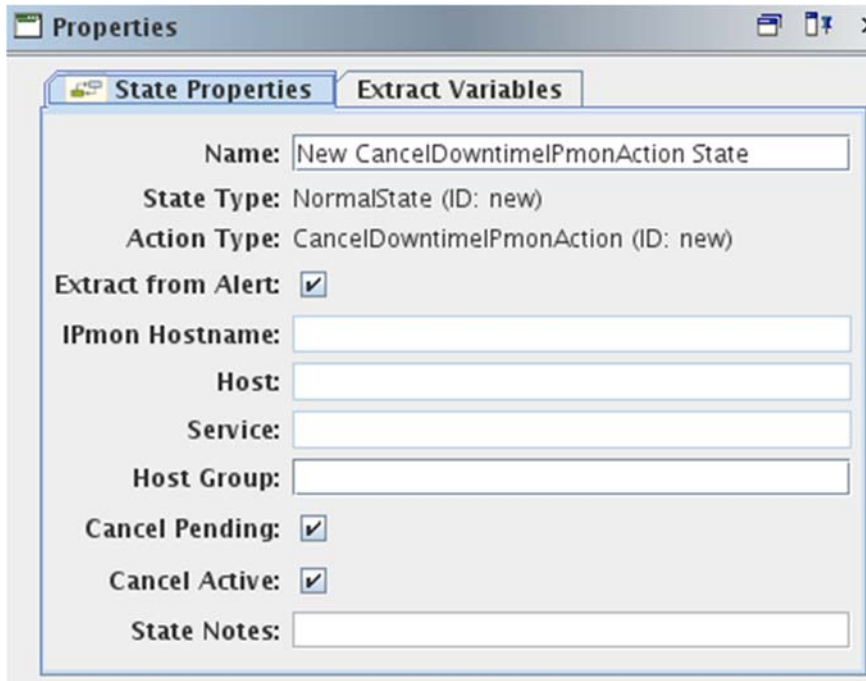
Downtime scheduled with this action starts immediately and continues "Duration" minutes into the future.

You **must** supply either a Host or Host Group.

Wildcards:

In the Host, Service, and Host Group fields basic wildcards are supported. For example, you can use '\*IPsoft\*' to match anything containing the word IPsoft. For this action, Host, Service, and Host Group are not case sensitive (thanks to [MySQL](#)), but you should not rely on this.

### Normal State - IPmon - Cancel Downtime



The screenshot shows a 'Properties' dialog box with two tabs: 'State Properties' (selected) and 'Extract Variables'. The 'State Properties' tab contains the following fields and options:

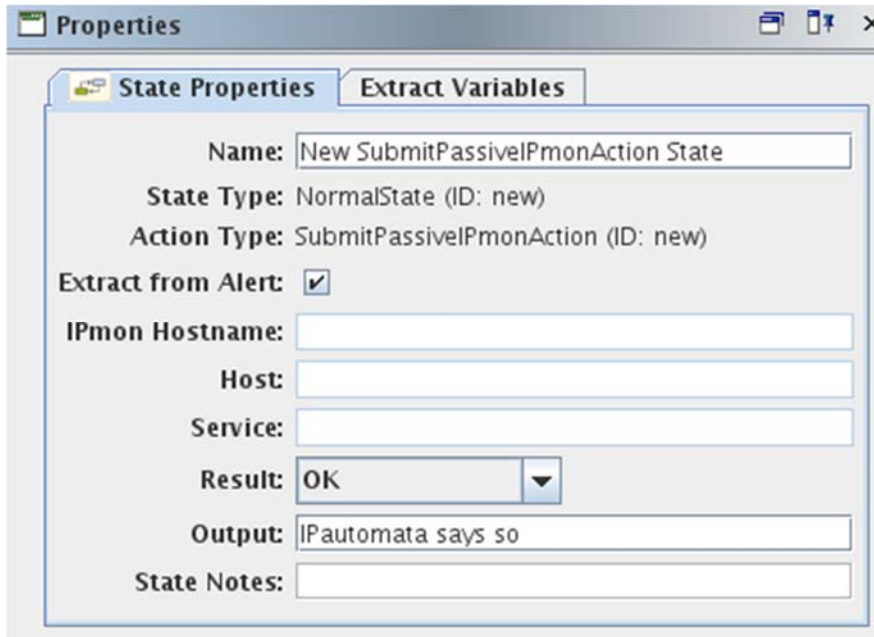
- Name:** New CancelDowntimeIPmonAction State
- State Type:** NormalState (ID: new)
- Action Type:** CancelDowntimeIPmonAction (ID: new)
- Extract from Alert:** ☒
- IPmon Hostname:** [Empty text box]
- Host:** [Empty text box]
- Service:** [Empty text box]
- Host Group:** [Empty text box]
- Cancel Pending:** ☒
- Cancel Active:** ☒
- State Notes:** [Empty text box]

The IPmon Cancel Downtime action is the exact opposite of the IPmon Schedule Downtime action. The return codes and matching fields behave the same.

The "Cancel Pending" checkbox allows canceling any pending (upcoming) downtime.

The "Cancel Active" checkbox allows canceling any active downtime.

### Normal State - IPmon - Submit Passive Result



**Properties**

**State Properties** | **Extract Variables**

**Name:** New SubmitPassiveIPmonAction State

**State Type:** NormalState (ID: new)

**Action Type:** SubmitPassiveIPmonAction (ID: new)

**Extract from Alert:** ☒

**IPmon Hostname:**

**Host:**

**Service:**

**Result:** OK ▼

**Output:** IPautomata says so

**State Notes:**

The IPmon Submit Passive Result action submits a single passive result to IPmon.

Possible Return Codes:

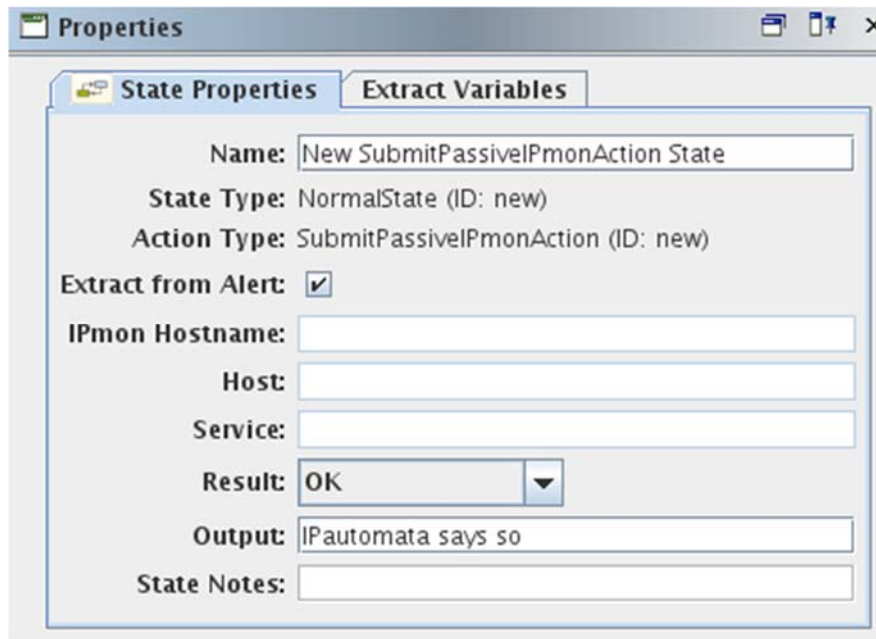
- 0 : OK
- 20 - Extract from alert set, but no ticket was associated with the execution
- 21 - Extract from alert set, but no IPmon alert associated with the ticket
- Anything else is an unexpected error

If the "Extract from Alert" checkbox is selected, the IPmon Hostname, Host, and Service values will be extracted from the IPmon alert associated with the radar ticket. If not selected, all three fields are required. Variables may be used with the `${var_name}` syntax.

Select the result to submit from the "Result" drop down and fill in the "Output" field (as if it came from a check).

### **Normal State - IPmon - Submit Passive Result**





The IPmon Submit Passive Result action submits a single passive result to IPmon.

Possible Return Codes:

- 0 : OK
- 20 - Extract from alert set, but no ticket was associated with the execution
- 21 - Extract from alert set, but no IPmon alert associated with the ticket
- Anything else is an unexpected error

If the "Extract from Alert" checkbox is selected, the IPmon Hostname, Host, and Service values will be extracted from the IPmon alert associated with the radar ticket. If not selected, all three fields are required. Variables may be used with the `${var_name}` syntax.

Select the result to submit from the "Result" drop down and fill in the "Output" field (as if it came from a check).

### Normal State - IPmon - Wait for Recovery

The IPmon Wait for Recovery action returns on IPmon alert recovery or the specified timeout, whichever is first. If the alert is already in the OK state, this will return immediately.

Possible Return Codes:

- -1 - Invalid Timeout String
- 0 - Recovered before timeout
- 1 - Timeout reached without a recovery
- 2 - No radar ticket or not an IPmon alert

- Anything else is an unexpected error

## IPdiscover - Discover device



The IPdiscover action, as its name suggests, runs IPdiscover in the specified device.

Parameters:

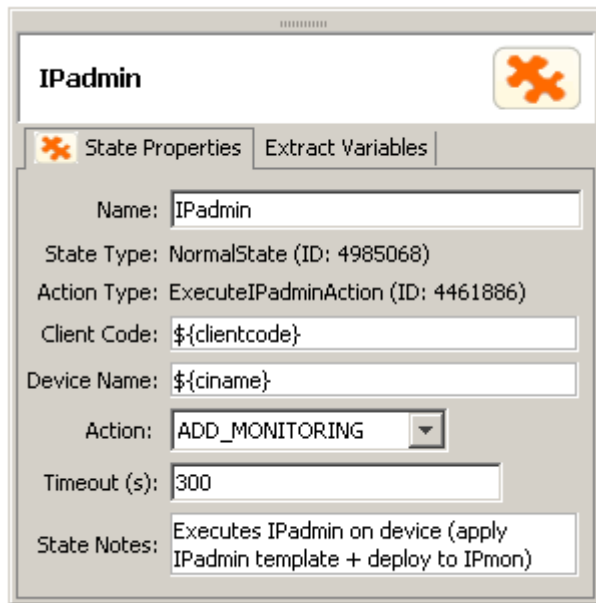
- **Client Code:** client code (short description, e.g. Acme)
- **Device Name:** name of the CI to be discovered.
- **IPmon Server Code:** name of the IPmon (NY1-IPmon11, IPmon08-NY1, etc). IPmon must appear in the list of assigned IPmons for this Client on IPdeploy.
- **Connections:** comma-separated list of the connections that will be used by the discovery process. See [IPdiscover: Assigning Connections](#) for more info.
- **Timeout:** amount of time (seconds) to wait for the discovery to complete. Please be generous when assigning a timeout, as the discovery process usually takes a few minutes to complete depending on several factors.

Possible Return Codes:

- 0 : OK
- 1: discovery process failed
- Anything else is an unexpected error

STDOUT contains the number of resources discovered.

## IPadmin - Apply template and enable monitoring



IPadmin action is designed to enable/disable the monitoring of a device.

Parameters:

- **Client Code:** client code (short description, e.g. Acme)
- **Device Name:** name of the CI to be discovered.
- **Action:** ADD\_MONITORING, REMOVE\_MONITORING.
- **Timeout:** amount of time (seconds) to wait for the discovery to complete. Please be generous when assigning a timeout, as the discovery process usually takes a few minutes to complete depending on several factors.

Possible Return Codes:

- 0 : OK
- 1: Error while applying the IPadmin template. Error description will appear in the STDERR

Output of the IPmon reload process appears in the STDOUT when the deployment is completed successfully.



user in another Phone Call Action, and finally that Phone Call Action would transition back to the initial one when its message has been spoken.

Possible Return Codes:

- -3 : Unknown error placing the call
- -2 : Error handling the response from the call
- -1 : Timed out waiting for a response
- 0 : OK if no prompt or if a multi digit prompt
- 1-9 : User single response

### HTTP Request State - Replacing curl

HTTP Request

State Properties

Extract Variables

Name:

HTTP Request

State Type:

NormalState (ID: 1953891)

Action Type:

HttpRequestAction (ID: 1648288)

Target URL:

http://ipcenter.ipsoft.com/IPportal/login.htm

Request Type:

POST

Request Body:

username=admin

password=....

Response Type:

HTML

Request Timeout (s):

5

HTTP Header Attributes

Override HTTP Headers:

Cookie VarName:

cookie\_1

Security

Enable Authentication?

☒

Request Type:

BASIC

Username:

authuser

Password:

authpassword

Secure Username?

☐

Secure Password?

☒

State Notes:

The HTTP Request action allows the system to make HTTP request to web servers. It is a replacement for the usage of curl and wget system commands, and the key features are:

- Support of both POST and GET HTTP methods.
- HTTP headers overriding
- Support for BASIC authentication
- Share the 'web session' between different states within the same execution via cookies.
- Ability to handle several web sessions within the same execution, by storing the cookies in different variables.

## Fields

- **Target URL:** URL the HTTP request will be sent to
- **Request Type:** [POST](#) or [GET](#)
- **Request Body:** Key/Value pairs with the request parameters (i.e. form input names and its values). One item per line.
- **Response Type:** HTML (will support TEXT in the future, by parsing the HTML document)
- **Request Timeout:** Timeout in seconds before the request is cancelled and an error is returned.
- **Override HTTP Headers:** you can override any of the HTTP Headers, such as the User Agent, Content Type or non-standard headers starting with the X- prefix.
- **Cookie Varname:** Execution variable name where the cookie will be stored. Usually if the website requires form authentication, you have to set a cookie variable name, and use the same variable name in the subsequent HTTP requests.
- **Enable authentication:** check this if the website requires HTTP authentication. Please note that HTTP authentication is not the same as Form authentication, so if the website requires the user to enter a username and a password in a web form, this option is not required. Only [Basic Authentication](#) is supported so far, [Digest access Authentication](#) will be added in the future.
- **Request Type:** Authentication type (BASIC, DIGEST will be available later)
- **Username:** username
- **Password:** password

## Outputs

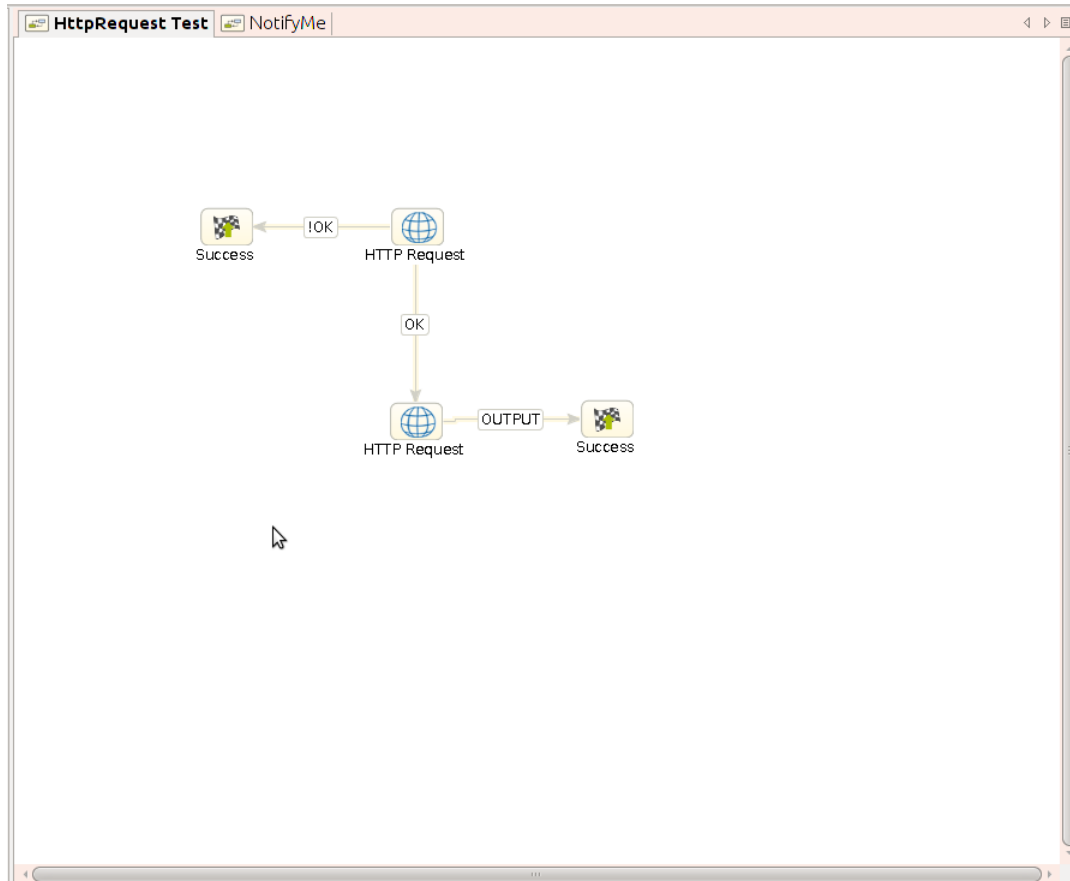
- Success: when the request is executed successfully (HTTP [Response Code 200](#)) the RET value is zero.
- Error: when the request returns an error (HTTP [Response Code <> 200](#)) the RET value is equal to the HTTP Response Code (1XX | 3XX .. 5XX).

In both cases, the full HTML document is stored into OUT.

**Use case scenario: executing a 2nd automata via IPcenter**

This automata will simulate an engineer that logs into IPcenter and executes another automata. The following steps will be executed:

1. The automata will log into IPcenter using the engineer's credentials, and doing a POST request to the IPcenter Login Form [<https://ipcenter.ipsoft.com/IPportal/login.htm>]
2. If the authentication succeeds, the automata will request the execution of another automata, by simulating a click on the Execute button on the [Automaton Details page](#).
3. Execution ends.



HTTP Request (1) Details:

- **Target URL:** <https://ipcenter.ipsoft.com/IPportal/login.htm>
- **Request Type:** POST
- **Request Body:** username=admin  
password=1p.....
- **Response Type:** HTML
- **Request Timeout:** 5
- **Cookie Varname:** cookie\_1

HTTP Request (2) Details:

- **Target URL:** <https://ipcenter.ipsoft.com/IPautomata/execute.htm>
- **Request Type:** POST
- **Request Body:** automaton=1027428  
runtimeVariables["']['phone']=5562
- **Response Type:** HTML
- **Request Timeout:** 5
- **Cookie Varname:** cookie\_1

### Wait State - Generic Event Listen

Generic Event Listener

State Properties

Extract Variables

Name:

Generic Event Listener

State Type:

WaitState (ID: new)

Action Type:

GenericTicketEventWaitAction (ID: new)

Match Type:

AND

▼

Timeout (min):


5

Matcher:

New

State Notes:





The Generic Event Listener state pauses the execution for up to the timeout duration specified, and on an incoming IPradar event checks the configured matchers. The state is configured by creating one or more event matchers. If you are using more than one matcher, the match type can be changed between AND and OR as desired. Click New to create a new matcher.

Possible Return Codes:

- -1 - Invalid Timeout String
- 0 - Match condition met
- 1 - Timeout reached without condition met
- 2 - No radar ticket or not an IPmon alert
- Anything else is an unexpected error

All matchers that matched will display the event type and the value of the matched ticket field in the Stdout in the following format:

Event Name: EVENT\_NAME\_1

Log Type: LOG\_TYPE\_1

attributeName\_1: attributeValue\_1

Event Name: EVENT\_NAME\_2

Log Type: LOG\_TYPE\_2

attributeName\_2: attributeValue\_2

Each event matcher has the following fields.

- **Event Type:** Required - The two most useful event types are IPMON\_UPDATE and UPDATE.
- **Log Action Type:** Optional - This is the Event subtype. This is what appears under "Type" in the radar ticket log view, and can be used to narrow the type of update to match on.
- **Ticket Field:** Optional - The bean path to the property from a [RadarTicket](#) that you want to run the expression on. This works the same as an Automata Ticket Matcher.
- **Expression:** Required if Ticket Field used, otherwise optional - Regex used on the Ticket Field to determine if this matcher has matched.
- **Negate?:** Invert match result
- **Do eventless checks?:** Eventless checks are done when this state executes and before the automaton sleeps. They are also done if the state times out. Certain matchers, like matchers on IPmon state of the check, should do eventless checks if it is possible for the event you are waiting for to already have happened. Because all eventless event matchers in this state are checked regardless of what event and log action type they are configured for, they should only be used where that behavior is desired.

Example: To wait for either the alert to recover or the client to respond, the following matchers would be used.

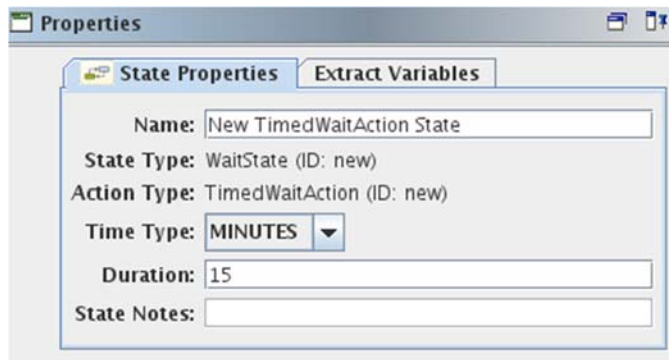
- Event Type: Update, Log Action Type: Correspond
- Event Type: IPMON\_UPDATE, Log Action Type: (blank), Ticket Field: ipmonTicketMapping.state, Expression: OK|UP

IPMON\_UPDATE is fired for updates from IPmon. These may make sense to have eventless checks run.

Here are other examples: <http://twiki.ip-soft.net/twiki/bin/view/IPsoft/GenericEventListenerExamples>

UPDATE is fired for [RadarTicket](#) updates. There is a Log Action Type for UPDATE events, which should be specified to narrow down the UPDATES the matcher will match on.

### Wait State - Timed Wait



The Timed Wait state pauses the execution for an arbitrary number of seconds, minutes, or hours.

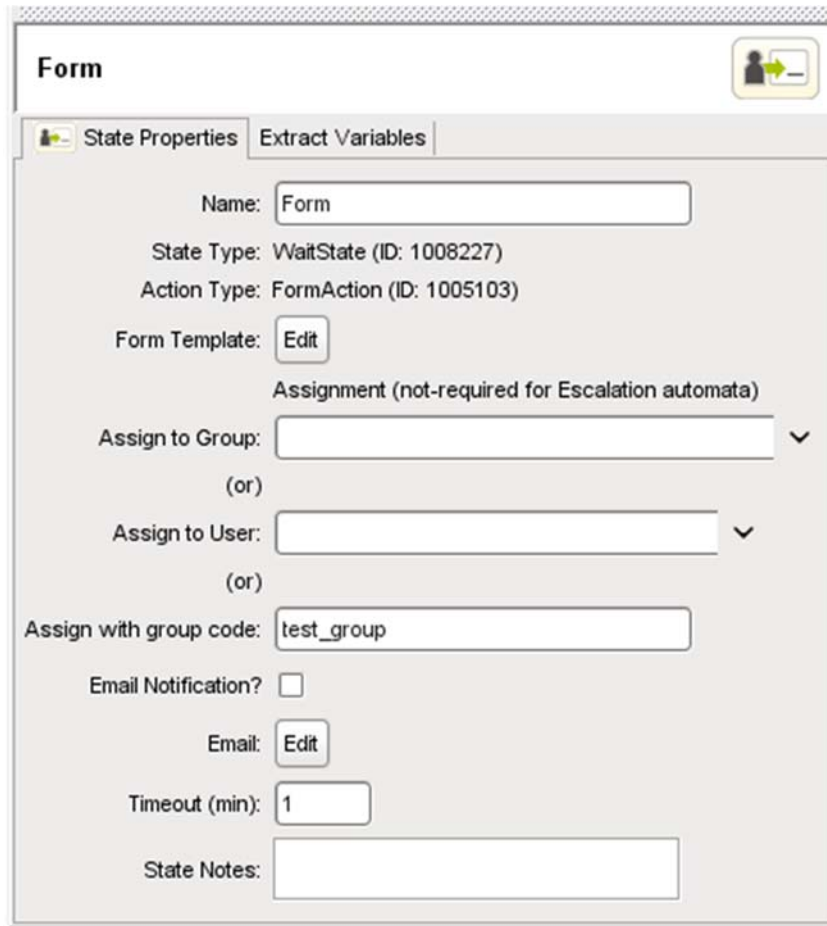
This state is typically better than executing a "sleep" command on the remote system. If you have to wait/sleep for over a minute, it is much better to use this state. Note, however, as with all wait states connections are closed upon entering the state. So, any state on the connection will be lost (ex current directory).

Possible Return Codes:

- -1: Invalid timeout specified, either not an integer or an unknown variable
- 0 : OK

Select the "Time Type" from the dropdown and enter the appropriate duration. Variables can be used for the duration via the `${var_name}` syntax.

## Wait State - Form Action



The Form Action allows capturing information (variables) from a user or set of users via an HTML form. Examples include requesting approval from a client before continuing the execution.

To edit the form, click the "Edit" button (see below).

A form can be assigned to a single user, or to all users with a specific role.

To assign to a specific user, simply select the user from the "Assign to User" drop down.

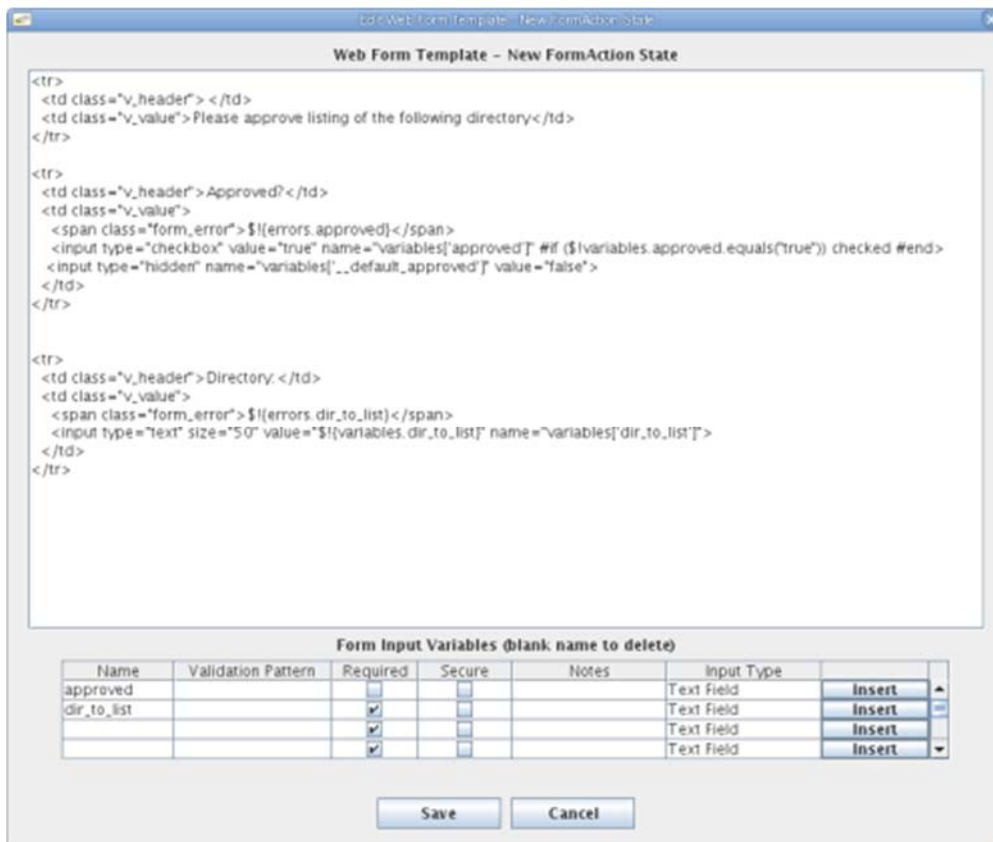
To assign to a group of users via a role, select the role from the "Assign to Role" drop down or enter the group code. Note that assigning to a group of users means that any one of the users in the role can complete the task. If the role does not yet exist, you must add the role and the associated members to the client through IPDeploy. If assigning to a group code, you may use `${}` variable syntax. The group is looked up on the execution automaton's client. If the group is not found, and the form is executing within a linked automaton, the group is then looked up on the inner automaton's client. This allows a form to be shared across clients via linked automata.

To notify users via email about task assignment, select the checkbox "Email Notification" and click the "Edit" button (see below).

Set the form timeout in minutes. If no response is given to the form within this time, execution continues and the return code of the form is 1. (If the timeout is not an integer or cannot be found in variable scope, the Form Action will return a code of -1.)

When a user successfully submits the form, the return code of the Form Action is 0.

Typically transitions out of a form action will be based on a variable collected in the form.



**Web Form Template - New FormAction State**

```

<tr>
<td class="v_header"> </td>
<td class="v_value">Please approve listing of the following directory</td>
</tr>

<tr>
<td class="v_header">Approved? </td>
<td class="v_value">
<span class="form_error"> ${errors.approved}</span>
<input type="checkbox" value="true" name="variables[approved]" #if ($variables.approved.equals("true")) checked #end>
<input type="hidden" name="variables[...default_approved]" value="false">
</td>
</tr>

<tr>
<td class="v_header">Directory: </td>
<td class="v_value">
<span class="form_error"> ${errors.dir_to_list}</span>
<input type="text" size="50" value="${variables.dir_to_list}" name="variables[dir_to_list]">
</td>
</tr>

```

Name	Validation Pattern	Required	Secure	Notes	Input Type	
approved		<input checked="" type="checkbox"/>	<input type="checkbox"/>		Text Field	Insert
dir_to_list		<input checked="" type="checkbox"/>	<input type="checkbox"/>		Text Field	Insert
		<input checked="" type="checkbox"/>	<input type="checkbox"/>		Text Field	Insert
		<input checked="" type="checkbox"/>	<input type="checkbox"/>		Text Field	Insert

Save Cancel

A form template is a [Velocity](#) template that generates snippet of HTML, basically the contents of a form tag. You do not need to worry about where the form is submitted. The content is enclosed in the following (logical) HTML:

<center>

<form>

<table>

<tr>

```

        <th colspan="2">State Name</th>

    </tr>

    --- Form Snippet Here ---

    <tr>

        <td colspan="2">

            <input type="submit" value="Submit"/>

        </td>

    </tr>

</table>

</form>

</center>

```

When creating a new form the initial content will only contain a single table row with a placeholder for the form description. Replace this content or remove it if not needed.

To make creating fully functional forms easy you can use the "Form Input Variables" section to create and insert the appropriate template text for different types of HTML input controls.

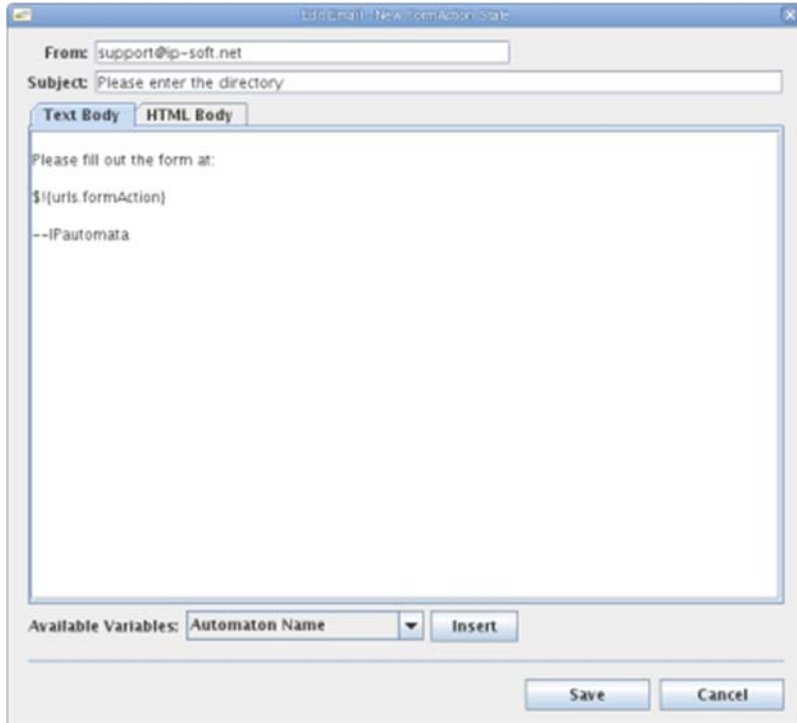
To add a new variable, simply fill out a row in the "Form Input Variables" section. To delete a variable, remove the row from the template text and remove it (make the name blank) from the "Form Input Variables" section.

The fields are:

- Name: The variable name
- Validation Pattern: A regular expression to be applied to the value. If it does not match a validation error will be presented to the user.
- Required: Whether the field is required
- Secure: Whether the variable is secure.
- Notes: Any notes about the variable.

To insert the variable, select the appropriate HTML "Input Type" and click "Insert". This will insert the appropriate template text at your cursor position. Edit the first table cell as necessary to label the form field. The default is the name of the variable. If inserting a select box or radio button group you must edit the

generated template text to set the appropriate values (replace val1, val2, etc.). There is currently no support for multi-select boxes.



From: support@ip-soft.net

Subject: Please enter the directory

Text Body HTML Body

Please fill out the form at:

\$!{url:formAction}

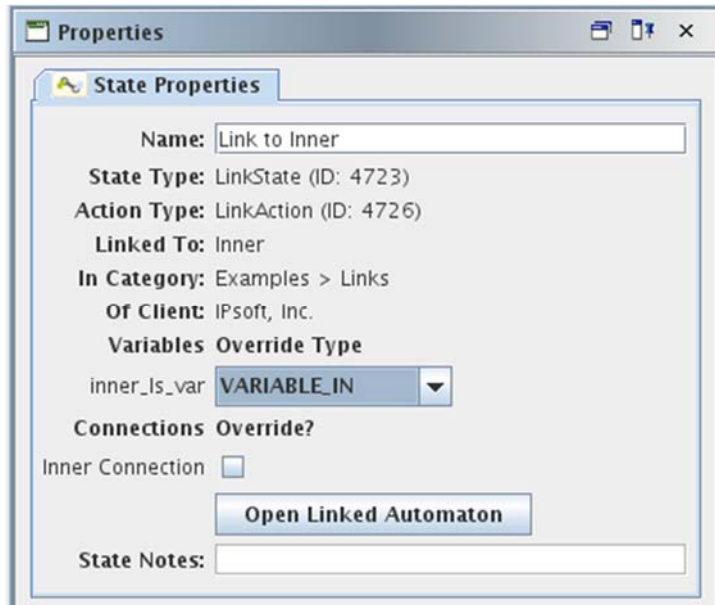
--IPAutomata

Available Variables: Automaton Name ▼ Insert

Save Cancel

Editing the notification email is nearly the same as editing an email for an Email Action without the option of hardcoding the recipient list. The most important variable to insert in the notification email is "URL - Form". Insert the variable by selecting it from the "Available Variables" drop down and clicking "Insert". When the email is sent the value of this variable will be a URL back to IPcenter for the specific form instance.

## Link State



A link state allows embedding an automaton within another automaton. When an execution reaches a link state it jumps into the linked automaton.

The variables of the target automaton are displayed and can be overridden with one of the following override types:

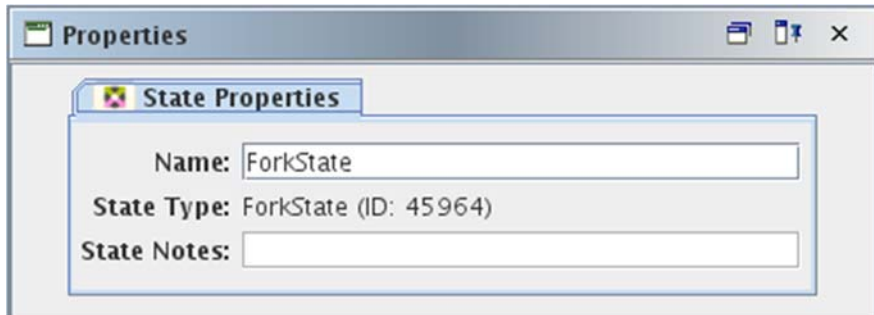
- **NONE:** No override at all, the variable will be available in the target automaton's scope only.
- **VARIABLE\_IN:** The variable from the outer automaton will override the value in the target automaton. The variable is passed "IN".
- **VARIABLE\_OUT:** The variable from the target automaton will be available in the outer automaton after the target automaton is executed. The variable is passed "OUT" of the target automaton. Note, you currently must have the variable defined in the outer automaton as well.
- **VARIABLE\_IN\_OUT:** A combination of both VARIABLE\_IN and VARIABLE\_OUT. The variable is passed "IN" to and "OUT" of the target automaton.

The connections of the target automaton are displayed and can be overridden. Selecting override will add a connection to the outer automaton with the same name (don't change it). You can change the connection information and the target automaton will use the connection from the outer automaton instead of its own connection.

**Do not attempt to recursion.** No, A->A, or A->B->A, etc. You can, however, have arbitrary levels of nesting and multiple links in an automaton.

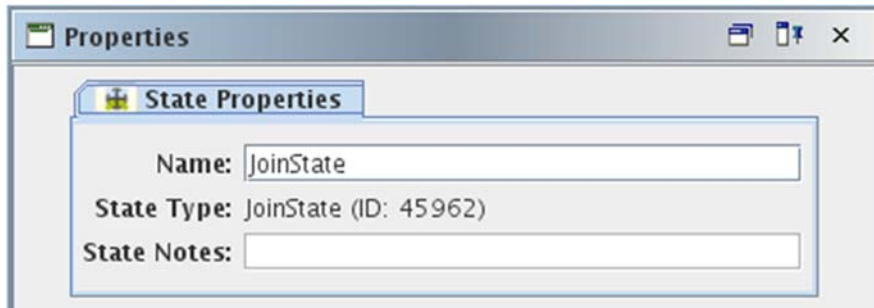
## Fork State





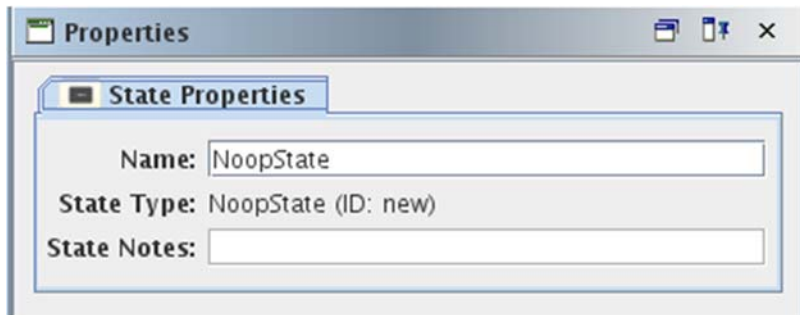
A fork state splits the execution into one or more concurrent paths. All transitions out of a fork must be of the ALWAYS type.

### Join State



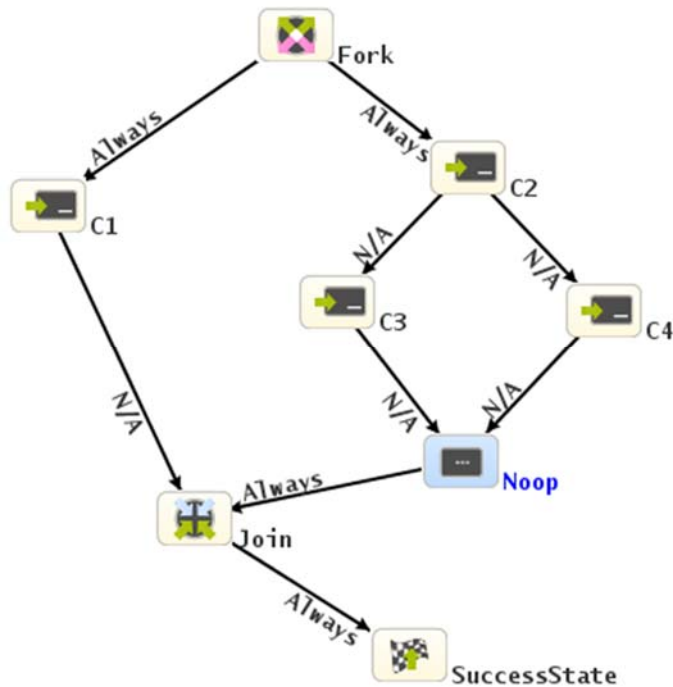
A join state joins one or more concurrent paths of execution. For execution to continue past a fork all incoming transitions must have been traversed.

### Noop State



A Noop state does nothing. It simply passes execution through a single outgoing transition. There are specific graphs that require a Noop state to join two branches together. For example, the one to the right requires

the Noop state to ensure that all incoming transitions to the join are traversed.



## Edit Variable Action State

**State Properties**

Name:

State Type: NormalState (ID: new)

Action Type: EditVariableAction (ID: new)

Variable:

Refresh: ☐

Methods:

Script:

Math:

State Notes:

An Edit Variable Action state allows changing the value of a variable at runtime.

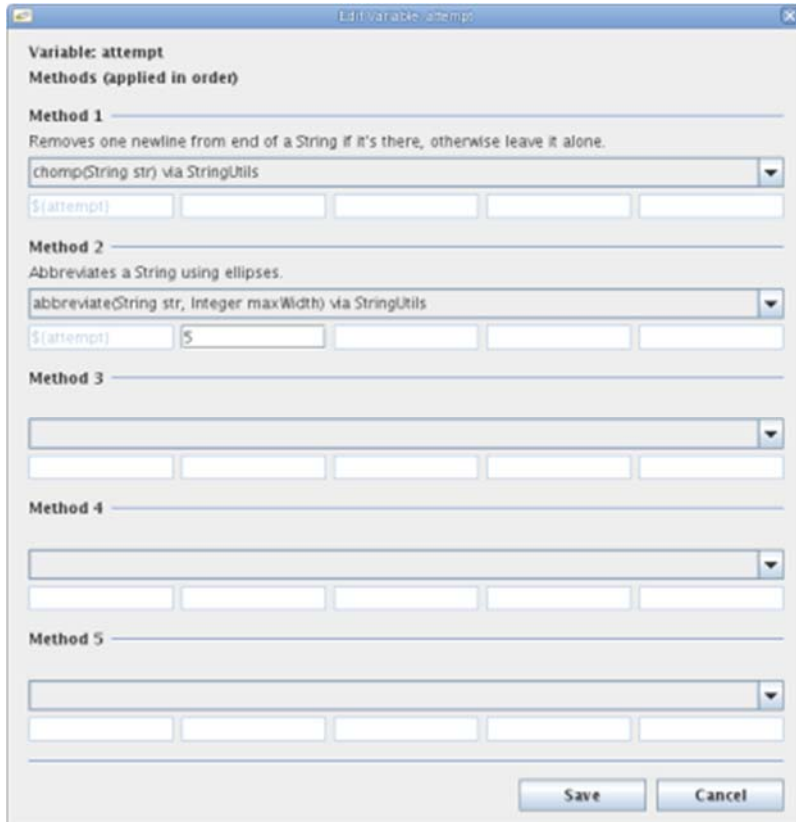
Select the variable to act on from the "Variable" dropdown.

The "Refresh" checkbox is only available for runtime variables. If selected, the runtime variable's value will be refreshed from the ticket.

There are three different types of edits/transformations available. Any combination of transformations can be selected. They are applied in the following order:

1. Methods
2. Script
3. Math

If you require a different order, create multiple Edit Variable Action states.



Variable: attempt

Methods (applied in order)

**Method 1**

Removes one newline from end of a String if it's there, otherwise leave it alone.

chomp(String str) via StringUtils

\$(attempt)

**Method 2**

Abbreviates a String using ellipses.

abbreviate(String str, Integer maxWidth) via StringUtils

\$(attempt) 5

**Method 3**

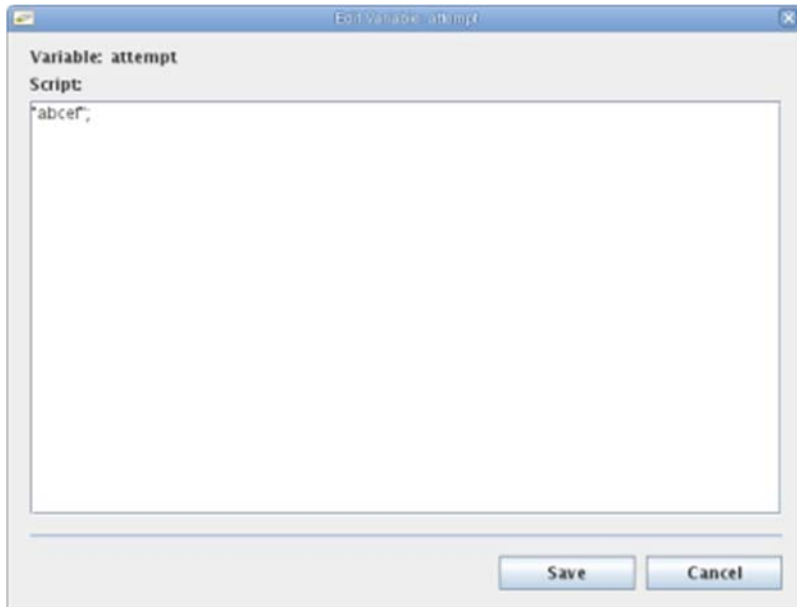
**Method 4**

**Method 5**

Save Cancel

Up to 5 method transformations can be configured. The transformations are applied in order.

Select the method name from the dropdown. The documentation for the method is presented above the drop down. The argument fields below the dropdown activate/deactivate based on the number of arguments the method accepts.



You can write arbitrary Javascript to modify a variable. The return value (last expression) of the script is evaluated as a string and assigned to the selected variable. All execution variables are placed into the script context and can be accessed as local Javascript variables.



You can use arbitrary mathematical expressions to transform a variable as well. Although it is possible to do similar things in Javascript, some details are handled better by using this type of transformation (you get 4 instead of 4.0).

For example, to increment the value of the "index" variable: '\${index} + 1'.

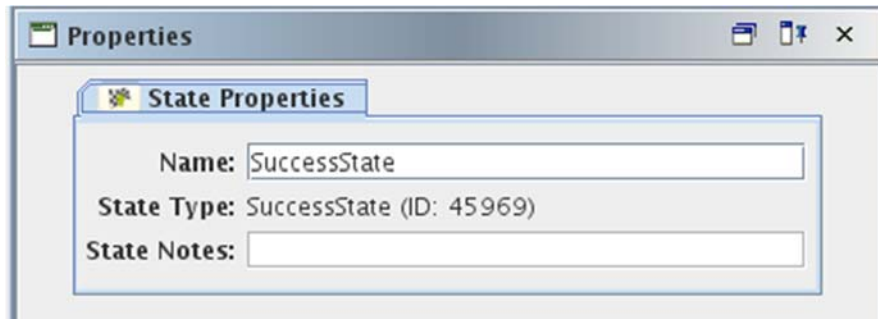
The expression syntax is that of JEP and is described [here](#).

**Caution:** JEP does not allow newlines inside of String variables. An automaton that uses JEP to compare Strings that contain newline characters will always fail. This can cause problems if you are replacing a variable with a value from an earlier execution, and that value had a newline For instance:

```
"${value1}" == "success"
```

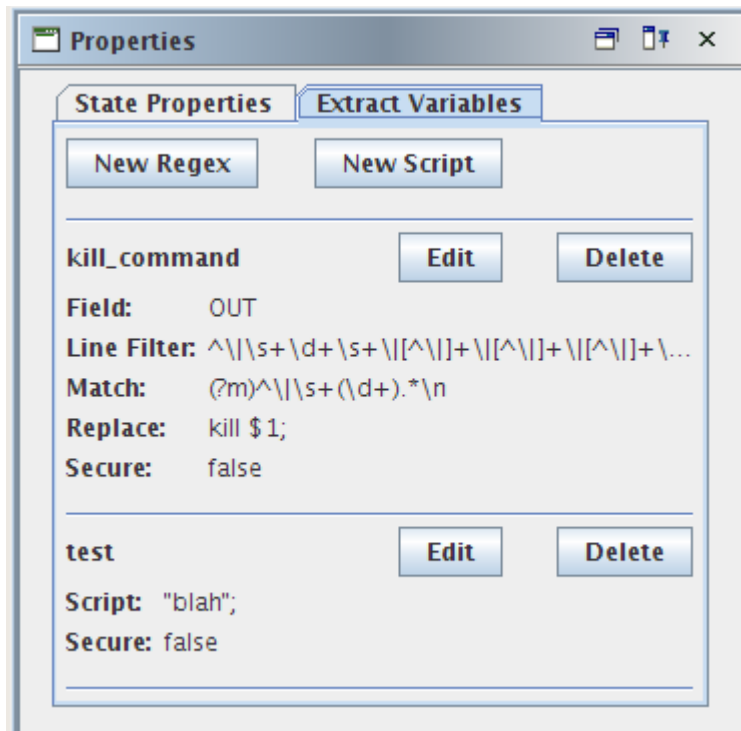
If the variable value1 contains any newlines, this expression will always fail. If you need to evaluate a String that contains newlines, do not use JEP, or preprocess it to remove newlines.

## Success State



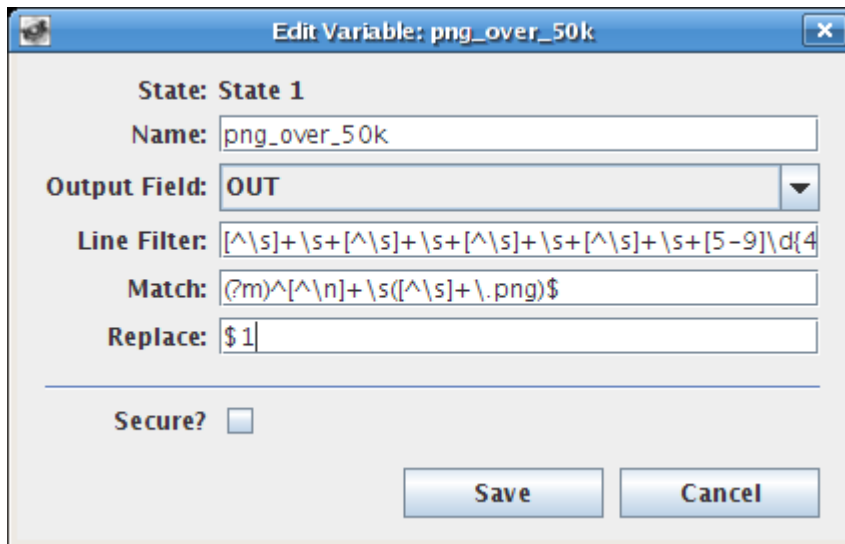
All automata require one or more success states. If the execution reaches a success state it is considered a success. If not, a failure.

## Extract Variables



Normal states allow one to extract variables from the output of the state. The "Extract Variables" tab lists all variables that are extracted from the state. Extracted variables are either "Regex" or "Script" variables, referring to how they extract their value.

## Regex Extract Variables



**Edit Variable: png\_over\_50k**

State: State 1

Name:

Output Field:

Line Filter:

Match:

Replace:

Secure? ☐

A regex extract variable extracts/manipulates data from the state's output.

Select the output field to extract the data from, out, err, or the return code.

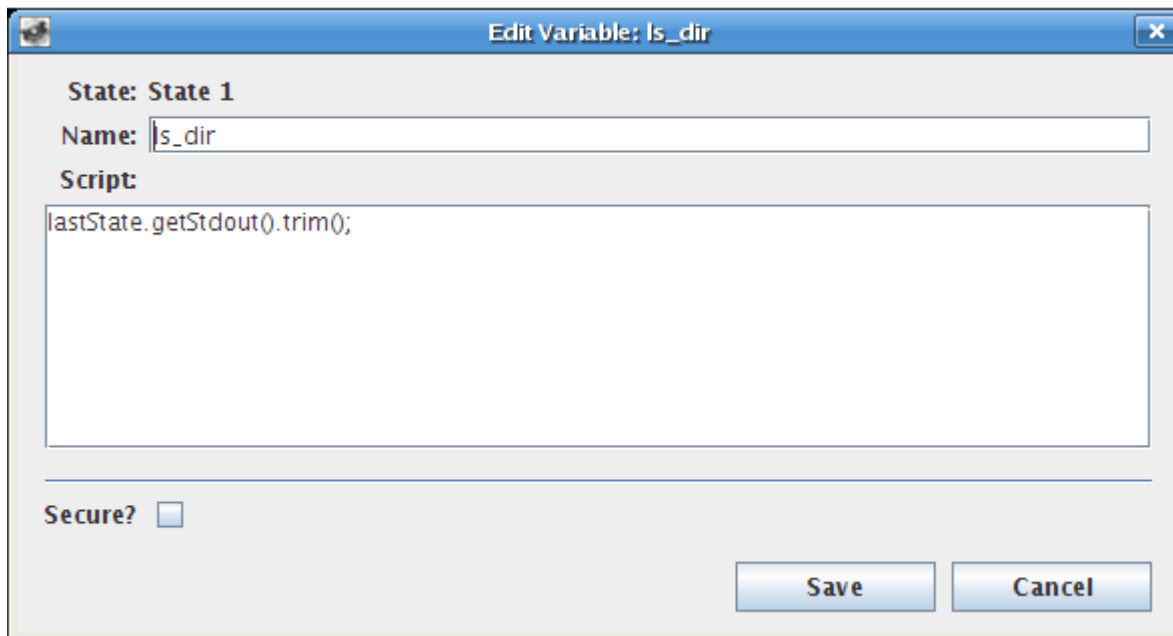
The line filter regular expression is applied to each line. Only lines that match the expression fully are passed on to the match expression and replace string.

The match expression and replace string apply to the output of the line filter to manipulate the data.

The example above takes the output of an 'ls -l' command and filters it for png files over 50k. The match expression and replacement string then extract the filenames.

You can test your line filter and match expressions with the [Regex Tester](#).

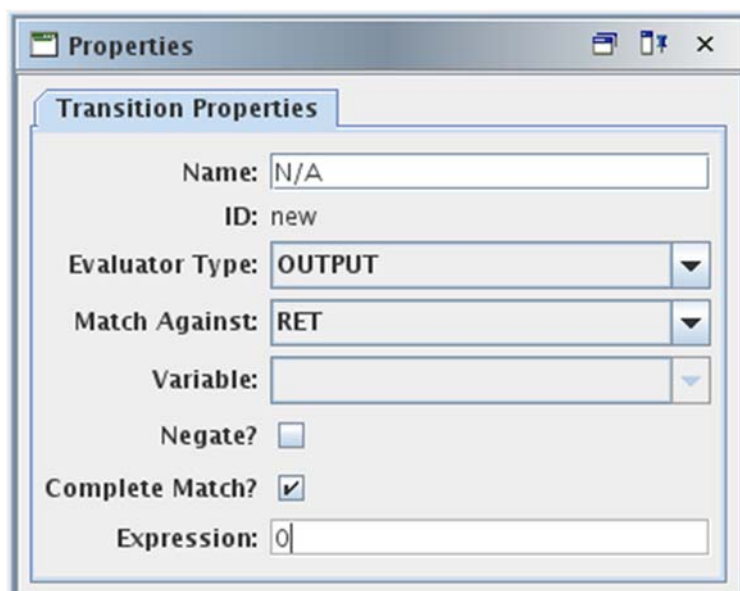
Script Extract Variables



Script extract variables allow arbitrary Javascript to examine the previous state's output and generate the variable's value. The previous state can be accessed via the "lastState" variable. The variable's value will be set to the script's value. The example above, simply returns stdout of the previous state, but much more complicated scripts are possible, although, not recommended unless needed.

Scripts can be tested with the [Script Tester](#).

## Editing Transitions



Select the transition (only) to edit the transition properties.

Give the transition a meaningful name.

If more than one transition evaluates to true, it is undefined which transition will be taken.

The following evaluator types are currently implemented.

- ALWAYS: This transition will always be taken.
- OUTPUT: This evaluator applies the regular expression against the field specified by "Match Against". For example, to match against the return code of the previous state, specify "RET" and set the expression to "0". This is the most basic type of transitions evaluator.
- VARIABLE: This evaluator applies the regular expression against the variable selected.
- JAVASCRIPT: This evaluator allows arbitrary Javascript as the transitions evaluator. All variables are placed in the context under their name along with the special variable "lastState" which is an instance of [StateExecution](#). You can access all properties of lastState. For example, lastState.getStdout() will give you the standard output of the last state. The script must evaluate to true or false. Anything but the boolean to 'true' is considered false. You can test your scripts with the [Script Tester](#).
- BOOLEAN\_EXPRESSION: This evaluator allows entering a boolean expression, as defined by [JEP](#). Comparing number ranges is significantly easier with this evaluator. For example: '\${index} > 20 && \${index} < 30'. In addition to normal execution variables, the return code, stdout, and stderr of the previous state are available under the JEP variables RET, OUT, and ERR respectively. Do **not** access these variables with the \${var\_name} syntax, but rather, just use the variable name. For example, 'RET > 0'.

Depending on the type of source state and action not all evaluator types will be available.

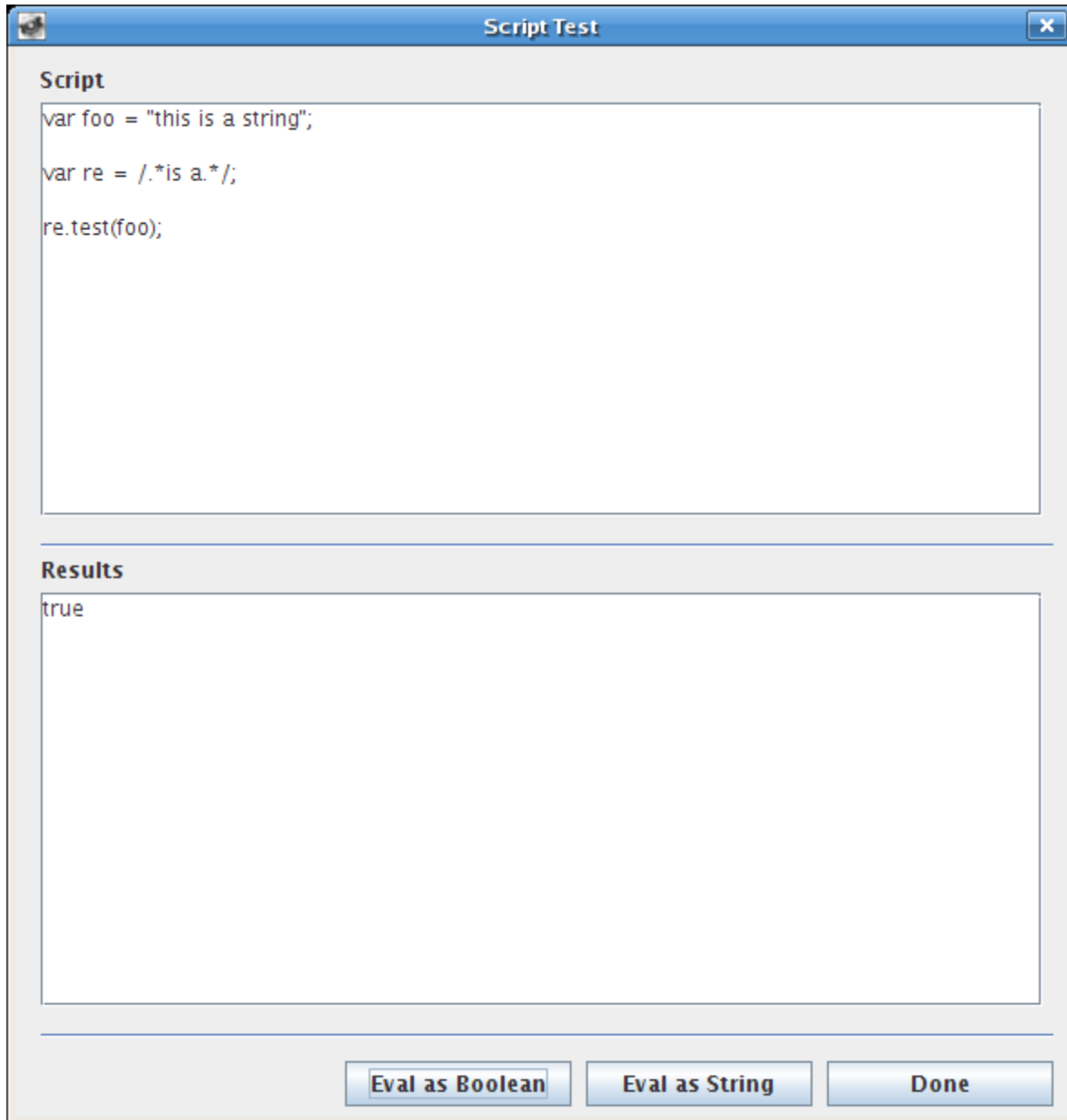
Checking negate simply negates the result of the evaluation. For example, a negated ALWAYS transition will be NEVER (no reason to do this one though...).

If complete match is selected, the expression must match the entire value. For return codes, this should almost always be true. If it isn't, the expression 0 will match a return code of 100 which is almost certainly not what is desired.

## 4.6 Testers

For non-trivial automaton it may be required to write moderately complex regular expressions and/or Javascript. To help in testing these, you can test your scripts and regular expressions manually via the appropriate Tester.



**Script Tester**

The Script Tester is accessible from the File menu. In the top text area you can write arbitrary Javascript and evaluate it as either a boolean script or as a string. Scripts are evaluated as booleans when used in a Transition and as strings when used as a Script Extract Variable.



## 5. Web GUI

### 5.1 Radar Widget

IPautomata

Available Automata

[+] IPcenter Apache Kill High CPU Processes

[-] IPcenter Apache Restart

Automaton Notes

Restarts IPcenter Apache

Use more specific automata if available.

Initial State

Change Directory

Command

cd /apps/apache/bin

Host

\${process\_host}

Runtime Variables

Name

process\_host

Value

prodipc02.ipsoft

Notes

Execute

Create New Automaton

Matching Automata Classes/Templates

Kill High CPU Processes

Edit

Create Automaton from Template

Automata Executions for Ticket

[-] IPcenter Apache Kill High CPU Processes

01/02 06:27

Success

Execution Complete

Execution History

View as Text

[+][+]

Name

Run Time

Status

[-]

Create Ticket and Post Alert

01/02 06:27:20 - 01/02 06:27:28

COMPLETE

Action Type: UpdateTicketAction

Command: Update Ticket

Return Code: 0

Stdout:

--Updated Ticket--

Radar Ticket: 2611501

Radar Queue: Ready Queue

Status: Active

IPpm Ticket: 488036

IPpm Queue: IPsoft-Ops

Stderr:

[-]

List Processes

01/02 06:27:28 - 01/02 06:27:28

COMPLETE

Action Type: HostCommandAction

Command: ps auxwww | sort -nr -k3 | grep '/apps/apache/bin/httpd' | grep -v grep

Connection: Monitored Host

Host: 192.168.19.112

Return Code: 0

Stdout:

```

apache      2289  50.2   5.4 116764 112064 ?        R    05:44   21:37 /apps/ap
apache      4410  49.5   1.5 36788 32116 ?        R    06:27   0:06 /apps/ap
apache      3684   1.5   2.1 49316 44804 ?        S    06:10   0:16 /apps/ap
apache      3298   0.7   2.1 49452 44892 ?        S    06:02   0:11 /apps/ap
apache      4367   0.4   1.4 34680 30028 ?        S    06:25   0:00 /apps/ap
apache      3432   0.4   2.0 46680 42340 ?        S    06:06   0:05 /apps/ap
apache      2843   0.4   2.4 53636 49452 ?        S    05:56   0:08 /apps/ap
apache      3973   0.2   1.8 42780 38212 ?        S    06:18   0:01 /apps/ap
apache      3386   0.2   1.9 44896 40412 ?        S    06:05   0:03 /apps/ap
apache      4409   0.1   1.4 33480 28848 ?        S    06:27   0:00 /apps/ap
apache      4406   0.1   1.4 33480 28848 ?        S    06:27   0:00 /apps/ap
apache      3969   0.1   1.7 41532 36928 ?        S    06:18   0:01 /apps/ap
apache      2901   0.1   1.8 43368 38784 ?        S    05:58   0:03 /apps/ap
root       17576   0.0   1.3 33480 28704 ?        S    00:00   0:11 /apps/ap
apache      4407   0.0   1.4 33480 28776 ?        S    06:27   0:00 /apps/ap
apache      4358   0.0   1.4 33800 29072 ?        S    06:25   0:00 /apps/ap

```

Stderr:

<

|||

>

[-]

Store High CPU Processes

01/02 06:27:28 - 01/02 06:27:28

COMPLETE

Action Type: HostCommandAction

Command: ps auxwww | sort -nr -k3 | grep '/apps/apache/bin/httpd' | grep -v grep | awk '{if (\$3 > 20) print \$2}' | sort -n

The IPautomata execution widget is visible on the IPradar pickup, update, and escalation pages. The widget is structured as follows:

### **Available Automata**

All automata that match this ticket (via their ticket matcher configuration). Clicking on the [+/-] button will expand the automaton and let you execute it. Text fields will show up for any required prompt variables. Clicking "Execute" will execute the automaton and the widget will refresh showing the progress until the execution is complete. Clicking on the automaton name will take you to details about that automaton and let you execute it outside the widget (but otherwise in the same way).

### **Matching Automata Classes/Templates**

All templates that match this ticket. You cannot execute a template, however, if you possess the appropriate privileges a "Create Automaton from Template" link will allow you to launch the designer and create a new automaton from the template.

### **Automata Executions for Ticket**

Shows all past automaton executions that are associated with this ticket. Clicking the [+/-] buttons expand executions and individual commands. Clicking the "View as Text" link will open a new window with the execution's history as text suitable for pasting into the ticket. Clicking other links will either take you to more detailed information about the execution and/or the automaton.

**The only action that will actually execute anything is the "Execute" button. Feel free to try the rest of the links to see what they do.**

## 5.2 Automaton Details

**Automaton Details for IPcenter Apache Kill High Memory Processes**

**Actions**  
[Open in Designer](#)  
[Past Executions](#)

**Ticketless Execution**  

Name	Scope	Value	Notes
process_host			
alert_text			

[Execute](#)

**Core Properties**

Name	Live	Archived	Template
IPcenter Apache Kill High Memory Processes	YES	No	No

Notes	Client
Kills IPcenter Apache processes using high memory. Use this for httpd memory alerts.	IPSoft, Inc.

Category	Execution Mode	Created	Modified	Modified By
IPcenter	AUTOMATIC	06/07/2007 11:59:16	12/14/2007 14:01:47	kmclaughlin (Kean McLaughlin)

**Connections**

Name	Host	Host Override Variable	Protocol	User	User Switch User	User Switch Method
IPmon Host	ipmon001.ny1		SSH			
Monitored Host	prodip001-ipssoft	process_host	SSH			

**Ticket Matchers (Match Type: AND)**

Ticket Field	Match Expression	Negate
description	IPcenter Production Webserver 'd+/Apache Daemon Memory usage	No

**Constant Variables**

Name ( Notes )	Secure	Value
rss_memory_threshold	false	950000
process_grep_expr	false	'/apps/apache/bin/httpd'
ipmon_service	false	Apache+Daemon+Memory+usage
wait_seconds ( Number of seconds to wait. )	false	15

**Runtime Variables**

Name ( Notes )	Secure	Prompt	Ticket Field	Match Pattern	Replace Pattern	Validation Pattern
process_host	false	false	ipmonTicketMapping.hostname			
alert_text	false	false	note			

**Regex Extract Variables**

Name ( Notes )	From State	Secure	Field	Filter	Match	Replace
----------------	------------	--------	-------	--------	-------	---------

Clicking the name of an automaton throughout IPautomata will take you to the automaton details page. That page displays all information about the requested automaton included a graphical representation of the state space. The Automaton can also be executed from this page, both with a ticket (if opened from the widget) or without a ticket if navigated to directly. All required variables will be prompted, including those that would normally come from the ticket if executing without a ticket. You can also view past executions for a given automaton and open the automaton in the designer from this page.

IPSOFT PROPRIETARY

58

## 5.3 Execution Search

**Execution Search**

**Status**  
 READY  
 RUNNING  
 COMPLETE  
 ABORT\_REQUESTED  
 ABORTED  
 FAILED

**Creator**  
 All

**Client**  
 All

**Automaton**  
 All

**Radar Ticket #**  
 [Empty]

**IPpm Ticket #**  
 [Empty]

**Created After**  
 01/01/2008 14:55:01

**Created Before**  
 [Empty]

**Success** **Refresh Every** **Pause** **Page Size**  
 Either 1 minute [ ] 20

**Search**

Generated: 14:55:08

**Execution Search Results** 1 - 20 of 130 (Next)

Success	Automaton	Client	Radar Ticket	IPpm Ticket	Status	Created	Finished	Creator
Yes	Load average spike checker	Martha Stewart Living Omnimedia	2613679		COMPLETE	01/02 14:50	01/02 14:51	
RUNNING	KLP course release (versioning update)	Kaplan Professional	2613452	400374 / KP-deployments-KP	RUNNING	01/02 14:48		Jaslee (Jason Lee)
Yes	Oracle Instance Size - Data Collection Client	Knight Ridder Digital	2613610	400415 / krd-database	COMPLETE	01/02 14:41	01/02 14:42	Jmoran (Juan Moran)
Yes	Gather Load Information -	Reuters	2613575	400407 / Reuters	COMPLETE	01/02	01/02 14:23	

The execution search page allows monitoring current executions as well as viewing past executions. By default only executions with a status of READY or RUNNING will be displayed. The page will be refreshed at the specified interval. Older executions can be found by adjusting the search parameters.

## 5.4 Execution Details

Execution 2005 details for Automaton Linux Server Process Information							
Automaton	Radar Ticket	IPpm Ticket	Success	Status	Created	Finished	Creator
Linux Server Process Information	2613079		Yes	COMPLETE	01/02/2008 12:25:02	01/02/2008 12:25:06	
Incomplete Execution Pointers							
Execution Complete							
Execution History <a href="#">[view as Text]</a>							
[ + ] [ - ]	Run Time	Name	Actor	Scope	Command/Details	Status	
[ + ]	01/02 12:25:02 - 01/02 12:25:03	System Info			uname -a	COMPLETE	
[ + ]	01/02 12:25:03 - 01/02 12:25:03	uptime			uptime	COMPLETE	
[ + ]	01/02 12:25:03 - 01/02 12:25:03	Processes			ps aux --sort=-pcpu	COMPLETE	
[ - ]	01/02 12:25:04 - 01/02 12:25:04	Number of Processes			echo "517"	COMPLETE	
	Connection: IPSOFT Host Host: 192.168.19.122 Return Code: 0 Stdout: 517 Stderr:						
[ + ]	01/02 12:25:04 - 01/02 12:25:05	Top			/usr/bin/top -b -n 1   head -30	COMPLETE	
	01/02 12:25:05 - 01/02 12:25:05	Success			Success	COMPLETE	
Execution Variables							
Name	Scope	Value					
ipsoft_host		ipmon01.ny1					
number_of_processes		517					
Execution Log							
Created Thread	Pointer ID	Scope	Description	State	Status	Failure Reason	
12:25:02 ipautomataAutomaticExecutionListenerContainer-1			Execution created				
12:25:02 IPautomata-Executioner-4	2522		Connecting to 'ipmon05.ny1.ip-soft.net:22' for connection 'ipmon01.ny1 / SSH'				
12:25:02 IPautomata-Executioner-4	2522		Connected to 'ipmon05.ny1.ip-soft.net:22' for connection 'ipmon01.ny1 / SSH'				
12:25:02 IPautomata-Executioner-4	2522		Connecting to '192.168.19.122:1023' for connection 'ipmon01.ny1 / SSH'				
12:25:02 IPautomata-Executioner-4	2522		Connected to '192.168.19.122:1023' for connection 'ipmon01.ny1 / SSH'				
12:25:03 IPautomata-Executioner-4	2522		Normal State Executed	System Info	COMPLETE		
12:25:03 IPautomata-Executioner-4	2522		Normal State Executed	uptime	COMPLETE		

The execution details paged is reachable from the execution search page (among other places). It contains all the details that the widget contains for an individual execution. You can view the output of all states, the status, runtime, view the log as text, etc.

The one additional useful piece of information is the execution log. The execution log lists important phases during the execution and any errors that were encountered during the execution. If a connection fails, for example, the error will be visible in the execution log.

## 5.5 Approvals

[My Tasks](#)
[Automata Search](#)
[Execution Search](#)
[Automata Designer](#)
[Approvals](#)
[Help](#)

### Automata Approval

Status: PENDING

ID	Name	Client	Submitter	Submit Date	Reviewer	Review Date	Approval Notes
1001996	blah	IPsoft, Inc.	kmclaughlin (Kevin McLaughlin)	10/26/2009 12:49			Please approve <a href="#">View</a> <a href="#">Approve/Reject</a>

### Reviewing Automaton [1001996]

Name: blah	Client: IPsoft, Inc.
Submitter: kmclaughlin (Kevin McLaughlin)	Submitted: 10/26/2009 12:49
Created: 10/26/2009 12:49	Creator: kmclaughlin (Kevin McLaughlin)
Modified: 10/26/2009 12:49	Last Edited By: kmclaughlin (Kevin McLaughlin)
Category: Kevin	Execution Mode: MANUAL
Purpose: DIAGNOSIS	Notes:

#### Approval Notes

Please approve

[Approve](#)
[Reject](#)
[View](#)
[Open in Designer](#)

The approvals web UI shows all automata filtered by approval status. When viewing pending automata, and the current user has the authority to approve automata, an approve/reject link is available in the right column. Clicking the link opens the approval page where the automaton can be approved or rejected. The functionality provided by this page is the same approve/reject functionality as in the approvals view of the designer.