ACT-R 7.6+ Environment Manual

Working Draft

Dan Bothell

Table of Contents

Preface

This document is a work in progress to describe the operation of the ACT-R Environment. The content is accurate, but may not cover all the components of the Environment. It may also make reference to sections or other documents which are not yet available. The hope is that although it is not yet complete, this working version will be of some use to ACT-R modelers.

This document was written to describe the Tcl/Tk based ACT-R Environment application. The new HTML/javascript based Environment that can connect through a node.js based bridge to ACT-R (found in the examples/connections/nodejs directory or included as an application with the standalone versions of ACT-R) works similarly for the tools that it provides, except for the "Load" buttons. Because javascript cannot access the location of files, when using those tools you must select files located in the appropriate subdirectory of the running ACT-R's tutorial directory. For the "Load ACT-R Tutorial Model" button you must first specify the particular unit directory number with the Unit entry box, and for the "Load ACT-R Tutorial Experiment Code" button you must use the lisp subdirectory of the running ACT-R's tutorial directory.

Introduction

The ACT-R Environment is a GUI written in Tcl/Tk which can be connected to ACT-R to help users with running, inspecting, and debugging models. It also provides a way to display the experiment windows created with the ACT-R AGI tools (described in the docs/AGI document). It connects to ACT-R using the remote interface (described in the docs/remote document) and most of the information displayed by the Environment uses the same ACT-R commands that are available to the user when using ACT-R. Thus, for the most part, it does not provide anything new for working with ACT-R. However, because it displays the information in separate windows and can update those windows as the model runs it can be a lot easier to use when debugging, and for some things, like the buffer traces and BOLD response predictions, it displays the data graphically instead of just textually.

Running the Environment

If you are using the standalone version of ACT-R then the Environment will be started automatically when you run ACT-R (following the instructions provided with your version of the standalone). If you are running ACT-R in a Lisp application from sources, then you will need to start the Environment explicitly. These instructions cover the basic operation of the Environment being connected to the only version of ACT-R running on the same machine. If there are multiple versions of ACT-R running on the machine or you would like to connect the Environment to ACT-R running on a different machine then you will need to consult the network configuration information section.

With ACT-R running you can use one of these methods to start the Environment.

1. Run the pre-built Environment application found in the environment directory of the ACT-R distribution for your OS ("Start Environment.exe" on Windows, "start-environment-osx" on Macs, and "start environment Linux" for Linux). That will open the Environment and connect it to ACT-R.

    a. If you get a warning about allowing network access you will need to allow/enable that because it needs a TCP/IP socket connection to communicate with ACT-R.

    b. If you get a message indicating that it had an error trying to connect to ACT-R try pressing the No button to try again. If that keeps happening and No always succeeds then you should change the option to make that the default behavior.

2. If you are using Mac OS X and get an error dialog indicating that the file is "damaged and can't be opened" the issue is probably due to security permissions and not actually a damaged file. If you have the option in System Preferences under "Security & Privacy" to set "Allow applications downloaded from:" to Anywhere, you can try that and then run it again. If it successfully runs, then you can change your preferences back to a safer setting. If that is not an option,

another alternative is to try the run-mac-environment.command script instead of the application itself. You may need to Control-click on the script and select Open the first time to run it, and if you get a dialog about the start-environment-osx application not being from an identified developer you will have to select Cancel and then open System Preferences, select Security & Privacy, and in the General section should be a button that says “Allow Anyway” which you will need to press before it will work.

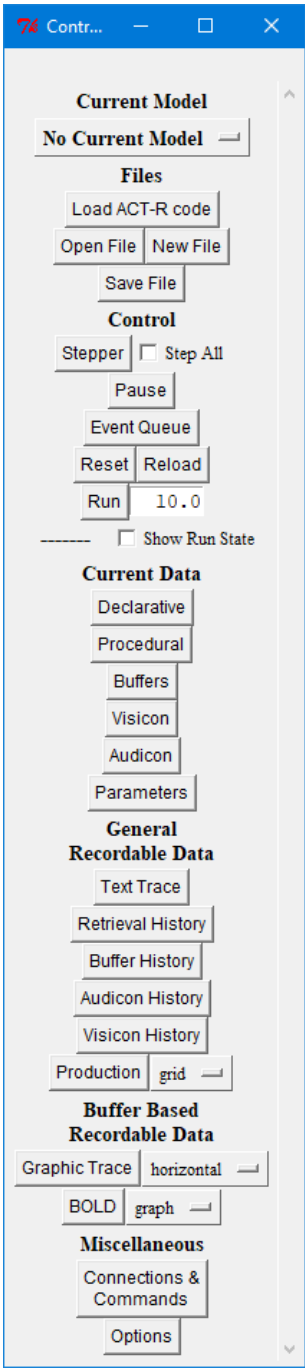
3. If you cannot or do not want to use the prebuilt applications then you can run the Environment from the source files. To do so you must have a Tcl/Tk wish interpreter installed (Linux and OS X often have it installed automatically). Then, all you need to do is either execute the starter.tcl script found in the environment/GUI directory or invoke it explicitly using wish i.e. "wish starter.tcl".

4. In some Lisps running ACT-R it is possible to start the Environment using the function run-environment. If the Environment does not start when you run that function or it displays a warning that run-environment is not available for your Lisp then you will need to use one of the methods above.

Once the Control Panel window opens and is filled with buttons the Environment is ready to use. When you are done using the Environment you can close the Control Panel window to disconnect it from ACT-R and close all of the open Environment windows.

Environment Overview

Once it is connected the Control Panel should look similar to the image below. The appearance will vary based on which operating system you are using, but all of the same components should be available.



The Control Panel consists primarily of buttons which open the tools that it provides. Those buttons are grouped into sections based on their functions. Each section of the Control Panel and its buttons will be described below. Note that there is a scroll bar on the right of the control panel and some of the buttons may not be visible without scrolling the window or making it larger when it is first opened.

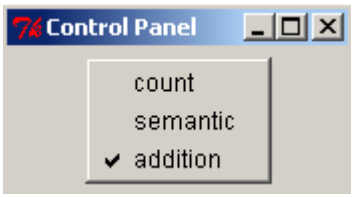
The Environment was originally designed to work with a single ACT-R model at a time, but it has been updated to allow all of the tools to work when there are multiple models currently defined. Most of the tools will be described as if there is only one model defined, but the details on using the Environment with multiple models can be found in the Current Model section.

If you encounter any problems with using the Environment please contact Dan (db30@andrew.cmu.edu) with the details.

Current Model

The Current Model section has only one item which is a button that serves two purposes. The first is to indicate which model is the current model. The text shown on the button displays the name of a currently defined model or the text "No Current Model" if there is no model currently defined. When there is a model name shown on the button all of the tools which require a model will be associated with that model, and the name of that model will be shown in the title of the corresponding tool's window. By default, if a model is deleted then all of the open windows which are associated with it will be closed, but that can be changed in the Options. One thing to note is that since most model files contain a call to clear-all loading a model file will close all the open model windows by default, but to avoid having to open all of the windows again while debugging a model, using the Reload button to load a model file again will not close all the windows associated with it.

The other purpose of the button is to change which model is the current one when there are multiple ones defined. If you press the button which shows the current model it will bring up a menu with all of the currently defined models' names in it. That would look like this if there were three models named count, addition, and semantic defined:



The one with the checkmark next to the name is the one currently being used, and clicking on one of the other names will switch that model to the current one in the Environment.

Files

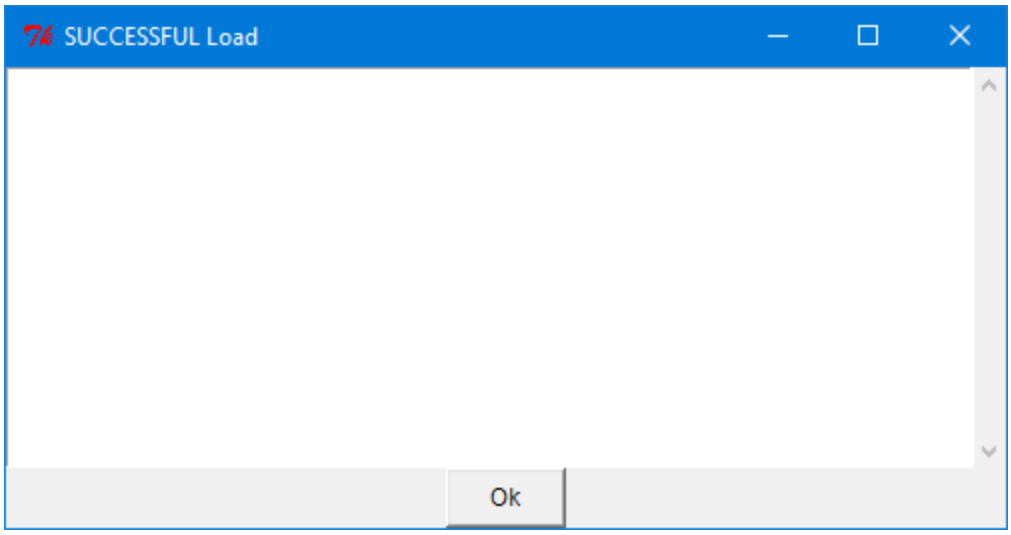
The files section contains controls for working with model files. It has buttons for loading a model file into ACT-R and provides a very simple text editor which one can use to edit model files.

Load ACT-R code

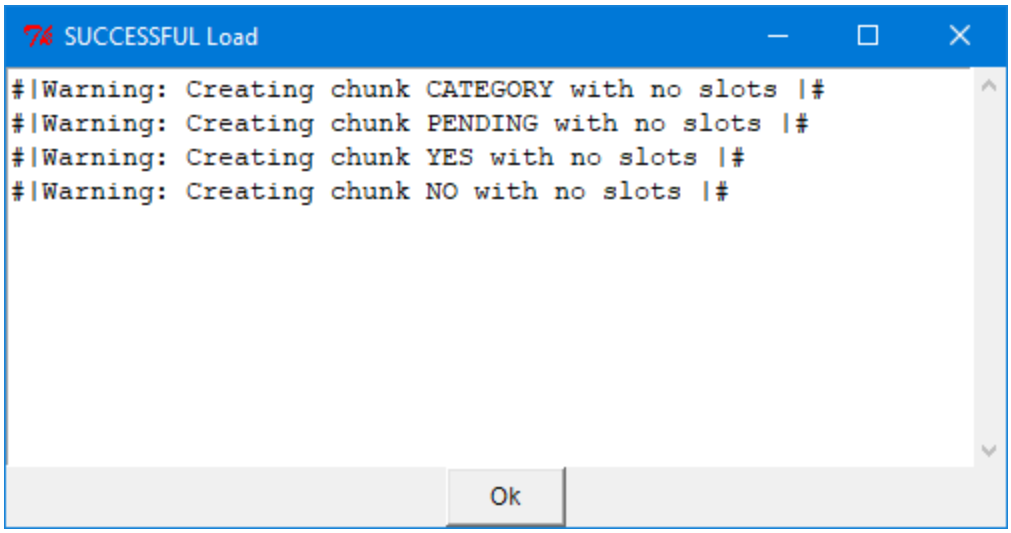
The "Load ACT-R code" button can be used to load a model file into ACT-R (or any Lisp file). This button will open a file selection dialog and the file which is chosen will be loaded into the Lisp running ACT-R.

If the compile option is enabled, then that file will be compiled and then loaded.

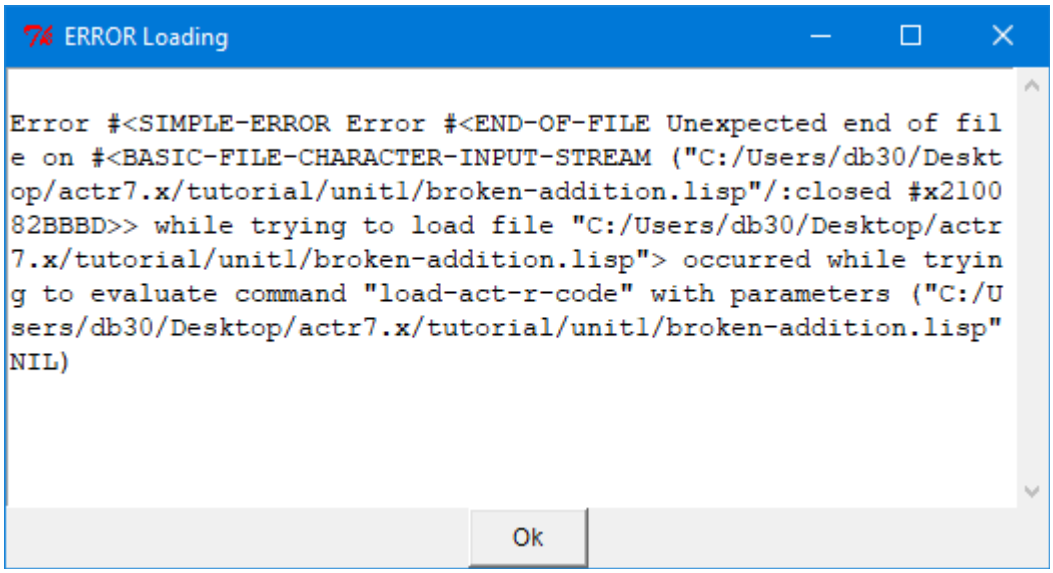
After loading the file a dialog window will be opened to show any output, warnings, or errors which occurred during the load. If the load completed successfully then the title of the window will be "SUCCESSFUL Load". The notification window may be empty if there were no warnings during the load:



or it may contain some warnings from ACT-R or Lisp:



If there is an error when loading the file the title of the window will be “ERROR Loading” and it will contain the error output which occurred:



In either case, the “Ok” button on the resulting dialog should be pressed to close the dialog before doing anything further with the Environment or ACT-R.

This button will only work if the Environment is running on the same machine as the Lisp running ACT-R. One thing to note is that if the Lisp you are using has a menu or other

easy to use mechanism for loading and compiling files then you should probably use that instead of this button because that is likely to provide much better handling of errors or other unusual circumstances.

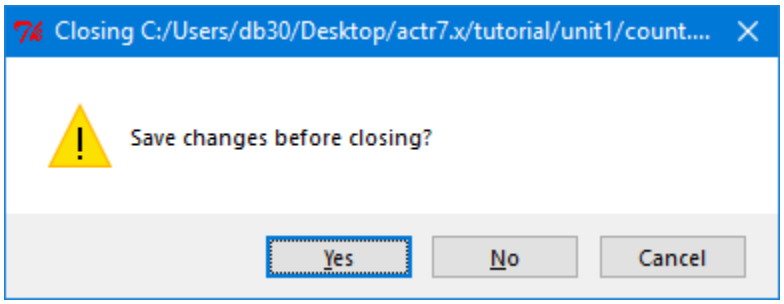
Open File

The “Open File” button will open a file selection dialog and then open the chosen file in a text editing window.

New File

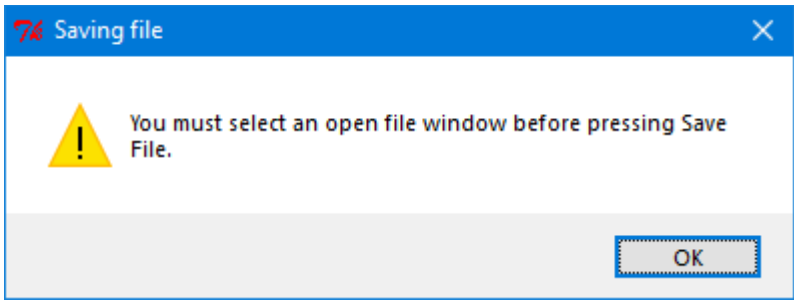
The “New File” button will open a file creation dialog to pick a new file to create and then open a text editing window to edit that file.

There may be multiple files open for editing through the use of those two buttons. If you try to close one which has been changed since it was last saved it will display a dialog to warn you about that and give you the option to save it immediately before closing the window, close the window without saving the file, or to not close the window:



Save File

The “Save File” button will save the contents of the currently selected edit window. If there is no open edit window or one is not selected then a dialog will be shown to indicate that this button has no effect:



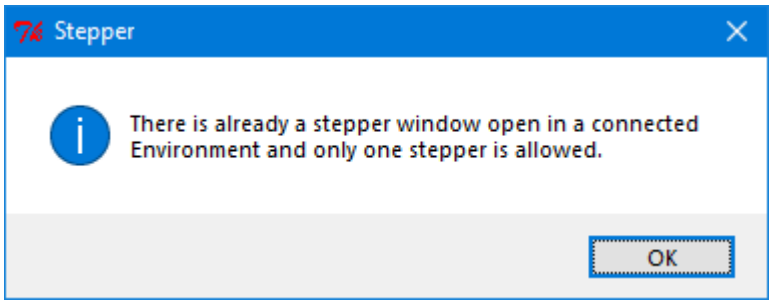
If the “Automatically save a backup” option is enabled, then before the file is saved a backup copy of the previous file will be made. Details on the backup files are described with that option, but the important thing to note is that the original file will always be the most recently saved version.

Control

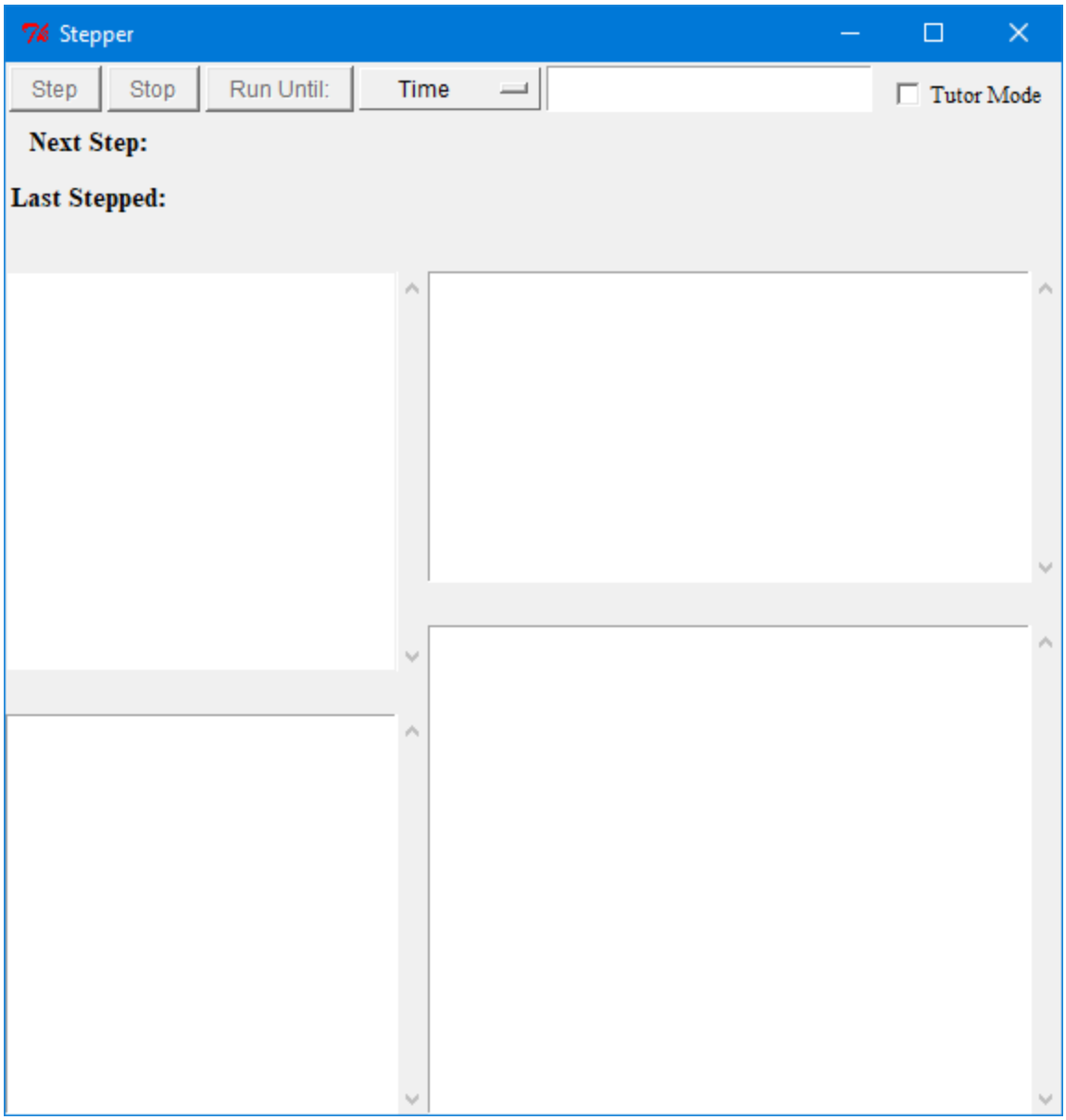
The control section contains buttons for stepping through the trace of running models, for viewing the events that are scheduled, running models, stopping running models, and for restoring the models to their initial conditions.

Stepper

The “Stepper” button is perhaps the most useful tool in the Environment. The Stepper is used to “step” ACT-R through its execution one event at a time. When the Stepper button is pressed it will open the Stepper tool if it is not already open. If it is open, then pressing this button will bring it to the front, but there can be only one Stepper open even when there are multiple Environments connected. If there is an open Stepper in another connected Environment then this warning will be displayed and no new Stepper opened:

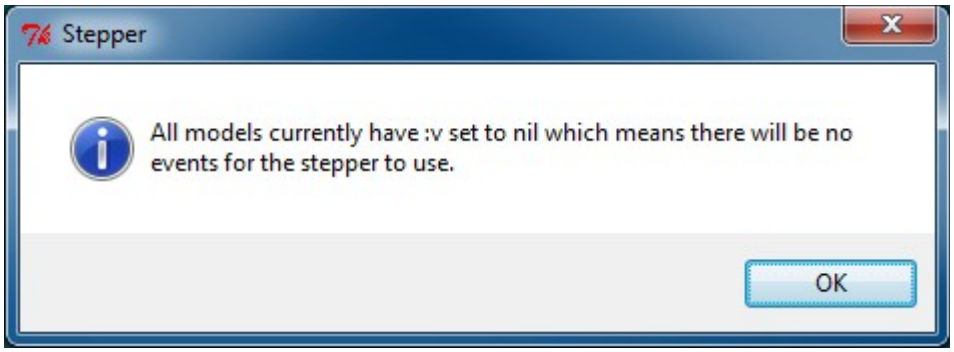


When the Stepper is opened it will look like this:



Each of the controls and displays will be described below.

When the Stepper is open, by default, it will pause ACT-R before every event that will be printed in the trace. That means if the :v parameter is set to nil in the model the stepper will not have any events on which to step. If all of the currently defined models have :v set to nil when the Stepper is opened it will display a dialog box like this one to warn about that:



That behavior can be changed using the "Step All" setting described below.

Next Step:

While the stepper is open and ACT-R is running it will display the event which will happen next after the "Next Step:" heading. ACT-R is suspended at that point and the modeler may use any of the Environment's other tools at that time to investigate the currently defined models. The model will execute that event once the user either presses the "Step" or "Run Until" button at the top of the Stepper (which are only usable while ACT-R is running) or closes the Stepper (closing the Stepper allows ACT-R to run to its normal completion).

Last Stepped:

When one of the buttons is pressed to step the model then the event displayed after "Next Step:" is moved down to the heading "Last Stepped:". Additionally, for some events, information will be displayed in the windows below that with additional details about the event that occurred before updating the "Next Step:" event.

Step

The "Step" button allows the system to continue operation. It will execute the "Next Event" which is displayed and ACT-R will continue to run to the next event which will be handled by the Stepper, if there is one. This is the button that is most often used with the Stepper – it "steps" the system through its operation one event at a time.

Stop

The “Stop” button will stop the current run before it executes the indicated “Next Step:” event. It is important to note that this button only stops the current run. If the system is being run from an experiment or other code which contains multiple calls to one of the ACT-R running functions then the system may continue to be run by the next call from that code after the “Stop” button is pressed i.e. the Stepper’s “Stop” button does not affect the code which may be providing an experiment or other interface for the model or models, it only stops the current instance of running ACT-R.

Run Until

The “Run Until:” button works in conjunction with the two interface items to its right which are a selection menu and a text entry box. Pressing the “Run Until:” button will execute the “Next Step:” event and allow the model to run without being paused by the Stepper until the condition specified by the combination in the selection menu and text entry box occurs.

There are three options for the selection menu: time, production, and module. That choice determines what should be entered into the text entry box. If an invalid value is entered into the text entry box when “Run Until:” is pressed then a warning will be printed in the trace indicating the issue and the system will be paused at the next available event as if the “Step” button had been pressed. Here are the details for specifying each of the options for “Run Until”.

Time

When “Time” is selected the text entry box should have a number entered into it which represents the ACT-R time in seconds when the Stepper should next pause the system. If that time has already passed then the model will pause on the next event as usual. If there is no event at that specific time, then the system will be paused at the first event after that time.

Production

When "Production" is selected the text entry box should have the name of a production entered into it. The system will then be allowed to run until the next time that named production is either selected or fired. If there are multiple models defined then the system will run until the next time any of those models selects or fires a production with that name.

Module

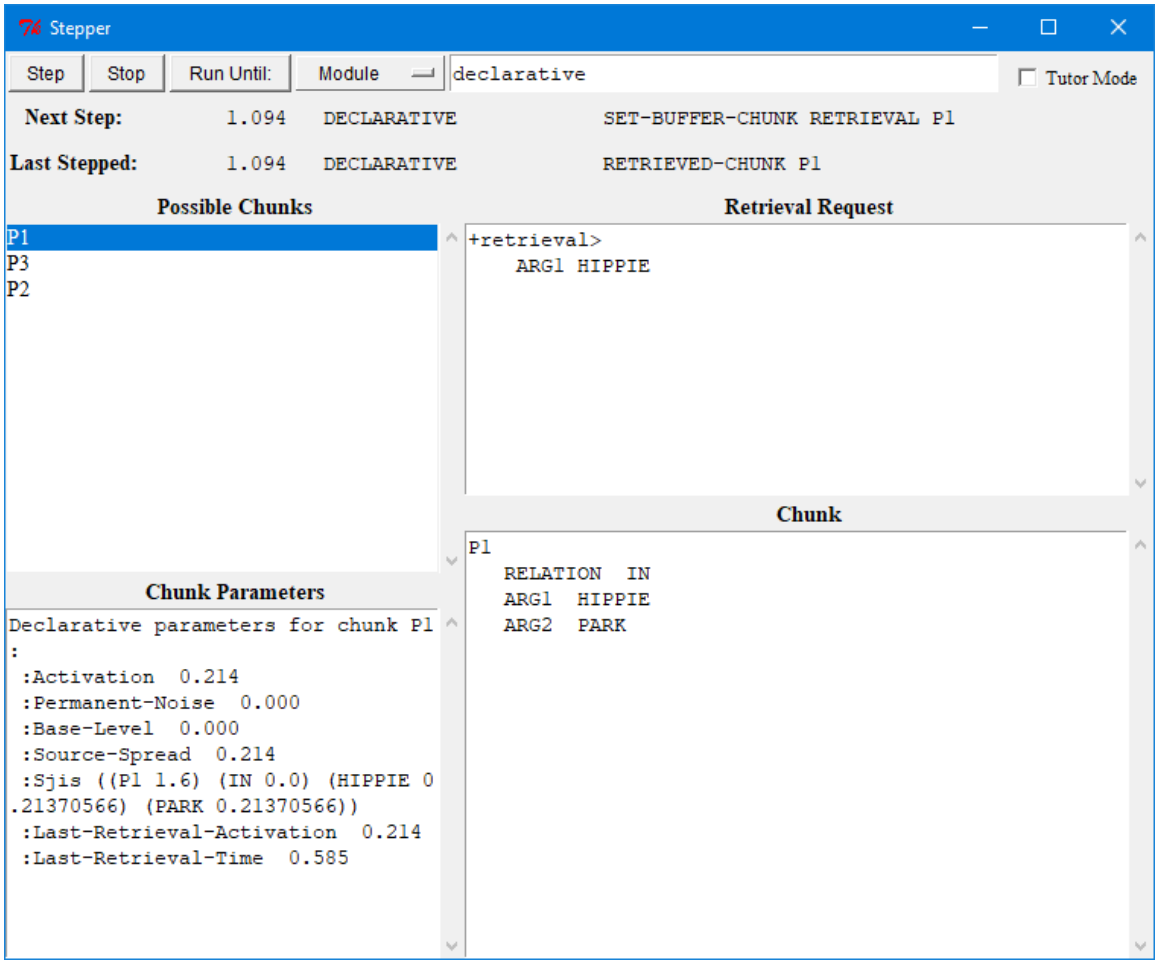
When "Module" is selected the text entry box should have the name of a module entered into it. The system will then be allowed to run until the next event which is generated by that named module. As with the production option, if there are multiple models defined then it will step to the first event generated by the specified module regardless of which model generated that event. It is important to note that it is the module name which must be specified (as shown in the trace) and not a buffer name. Some modules have the same name as their buffer, like goal and imaginal, but others do not, for example the module which controls the manual buffer is named motor.

Additional Information windows

The bottom portion of the stepper will show detailed information for three specific events within a model: the declarative module's retrieved-chunk event, the procedural module's production-selected event, and the procedural module's production-fired event. For all other events the detail windows at the bottom of the stepper will be blank.

Retrieved-chunk

When a retrieved-chunk event occurs the stepper will fill in the lower boxes like this (taken from a run of the fan model from unit 5 of the ACT-R tutorial):



The upper-right pane, labeled “Retrieval Request”, shows the request which was made to the declarative module that initiated this retrieval.

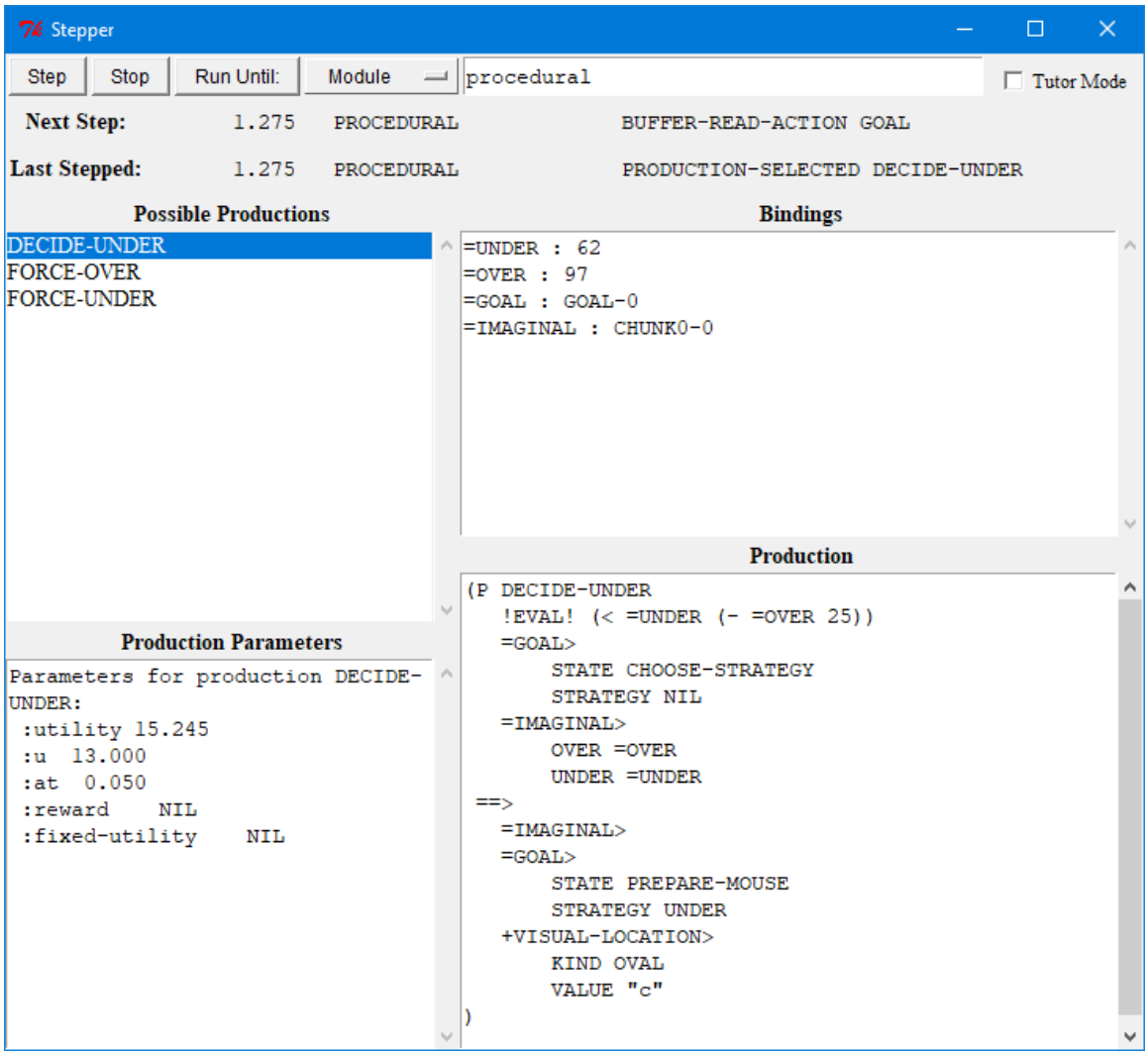
The upper-left pane, labeled “Possible Chunks”, shows the list of chunks which matched the request. They are ordered based on their activations from the highest at the top to the lowest on the bottom. Therefore, the chunk which is being retrieved is at the top. Selecting a chunk in this list will cause the lower two panes to be filled with the information appropriate for that chunk.

The lower-right pane, labeled “Chunk”, displays the standard ACT-R printout of the selected chunk. The lower-left pane, labeled “Chunk Parameters”, will be empty unless the subsymbolic computations are enabled for the model. If they are enabled, then that pane will display the declarative memory parameters for the selected chunk as reported by

the **sdp** command.

Production-selected

When a production-selected event occurs and the "Tutor Mode" box is not checked (see below for details when it is) the Stepper will look like this (taken from a run of the building sticks task in unit 6):



The upper-left pane, labeled "Possible Productions", shows the list of productions which matched the current state during the last conflict-resolution event. They are ordered based on their utilities with the highest utility, and thus the production which was selected, at the

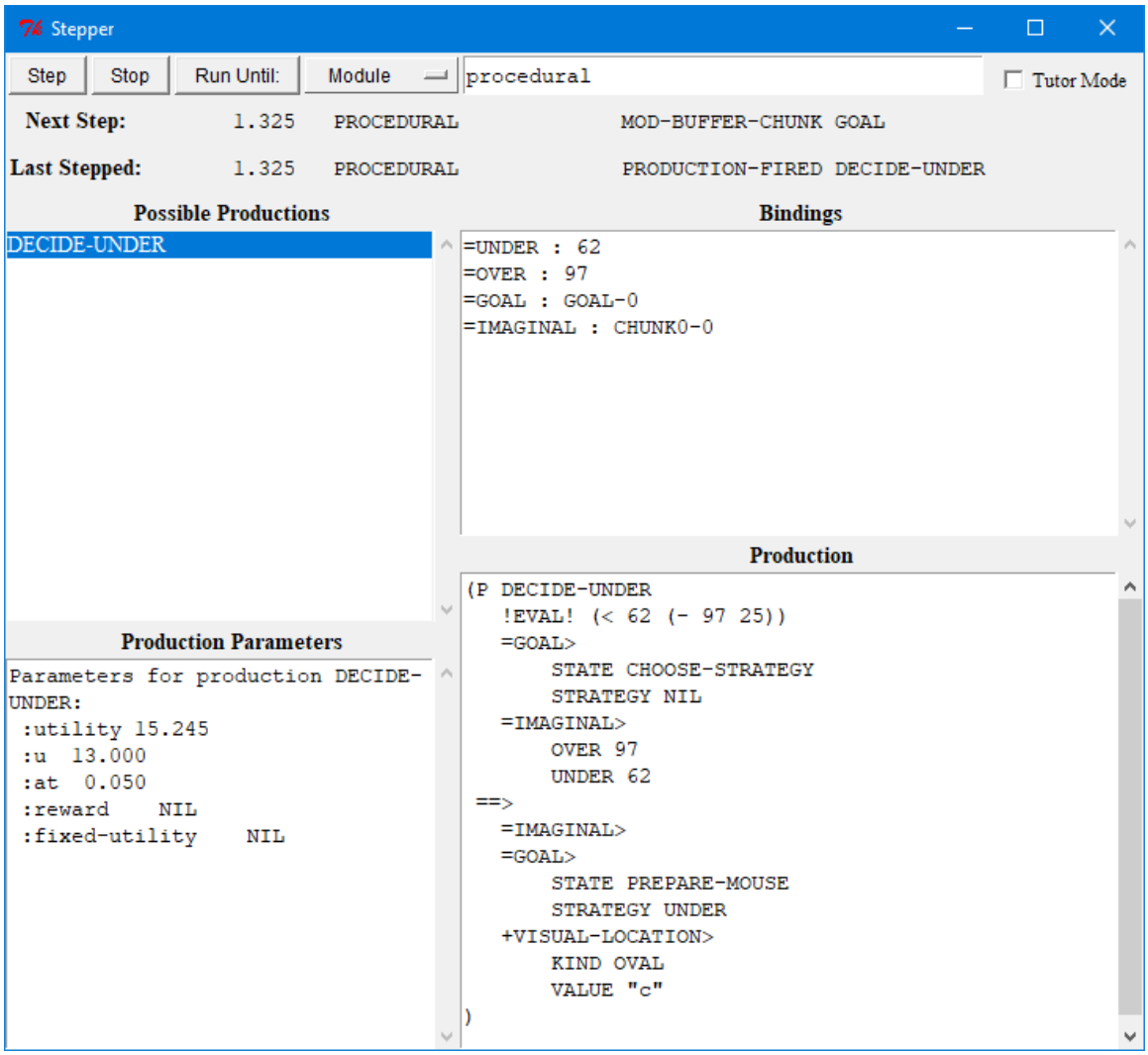
top. Selecting a production in this list will cause the other panes to be filled with the information appropriate for that production.

The lower-right pane, labeled “Production”, displays the text of the production. The lower-left pane, labeled “Production Parameters”, will be empty unless the subsymbolic computations are enabled for the model. If they are enabled, then that pane will display the procedural parameters for the selected production as reported by the **spp** command.

The upper-right pane, labeled “Bindings”, shows all of the variables used in the production and the value that they have while matching the current state.

Production-fired

When a production-fired event occurs the stepper will look like this (which is the production-fired event that follows the production-selected event shown above):

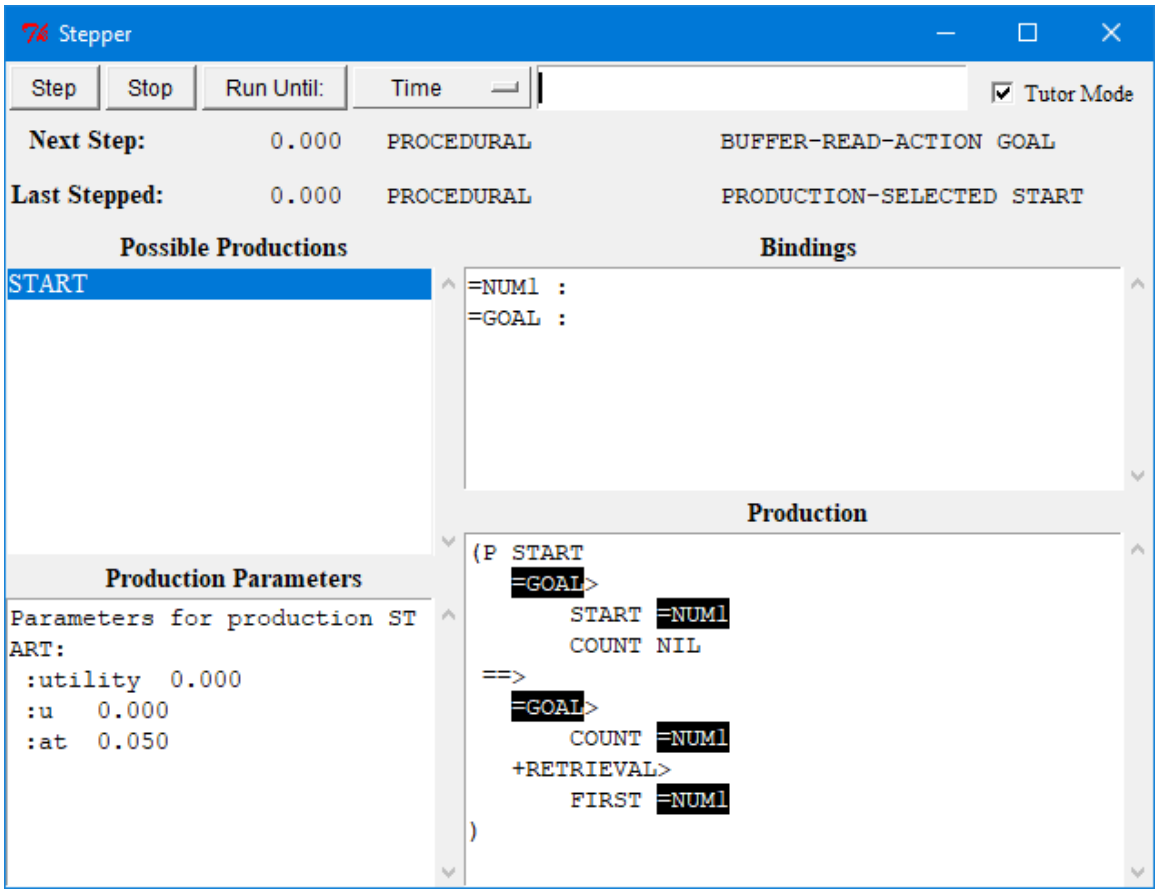


The displays are similar to those for the production-selected event. Now however, only the production which fired is listed in the upper-left pane. The "Bindings" and "Production Parameters" panes display the same information for that production which they did for the production-selected event. The information in the lower-right pane differs in that now it shows the instantiation of the production instead of the production text. The production's instantiation displays the production text with the variables replaced with their corresponding bindings.

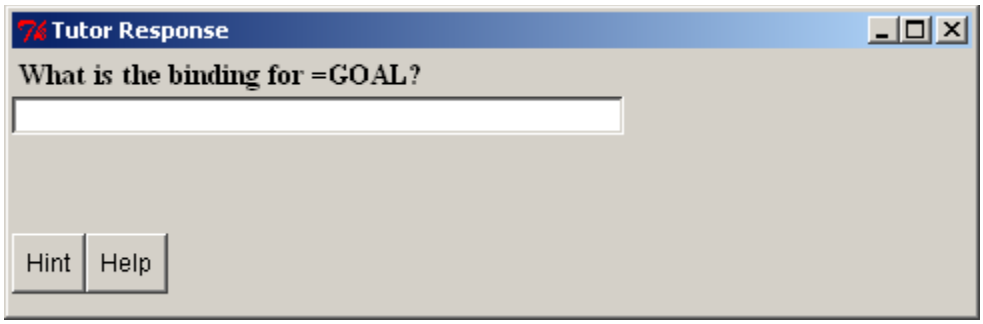
Tutor Mode

The "Tutor Mode" check box is for use primarily with the models in unit 1 of the ACT-R

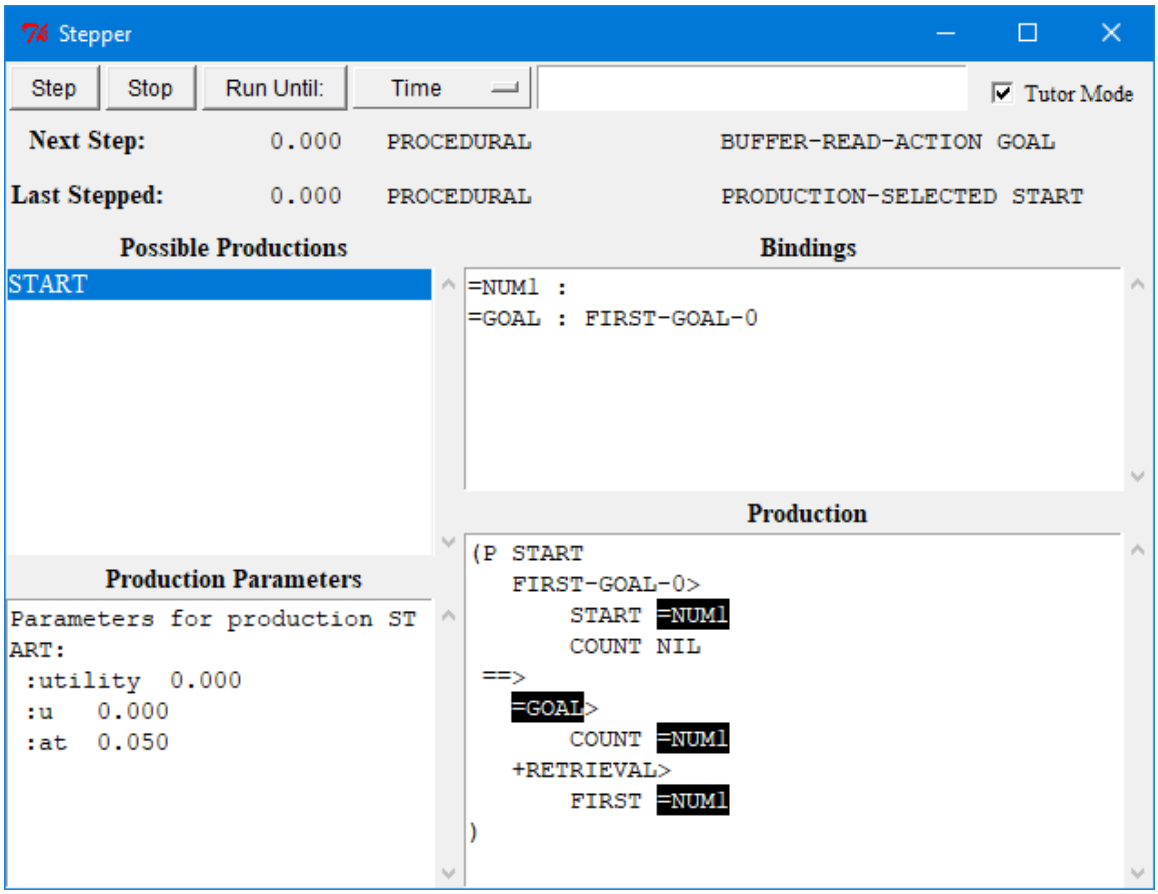
tutorial. When the box is checked the Stepper requires additional interaction from the user to continue past a production-selected event. Instead of displaying the production and its bindings for such an event the production is displayed with all of its variables highlighted and the bindings not shown like this (from the count model in unit 1 of the tutorial):



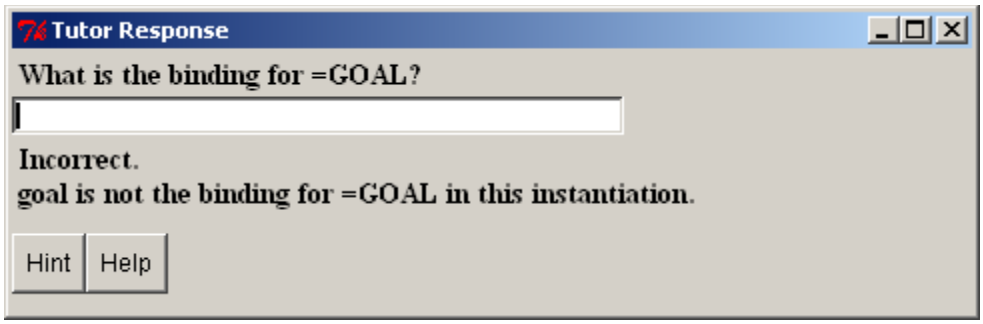
The user is then required to click on each of the highlighted variables and enter the appropriate binding. When one of the variables in the "Production" pane is clicked a new dialog opens in which the binding should be entered:



If the correct value is given then the “Tutor Response” dialog will close and it will replace the variable in the display and the value will be shown under the bindings:



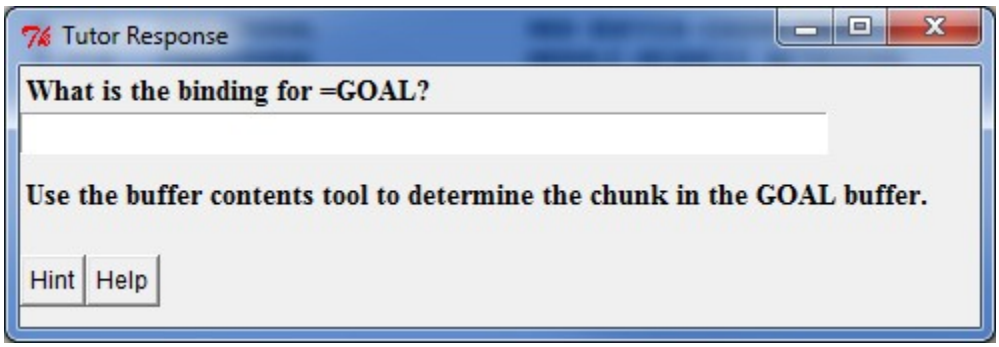
If an incorrect answer is given then it will indicate that the entry is incorrect and wait for another value to be entered:



The two buttons at the bottom of the "Tutor Response" dialog will provide additional information to help the user get the correct answer.

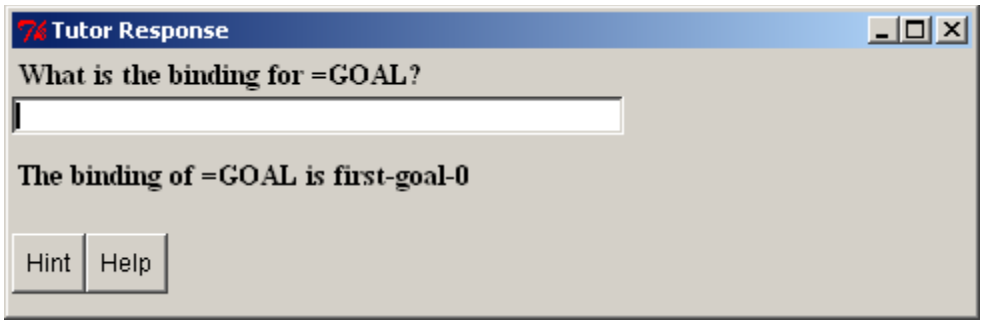
Hint

Pressing the "Hint" button will display a suggestion in the "Tutor Response" dialog indicating which other tool in the Environment may be used to help find the correct answer:



Help

Hitting the "Help" button will print the correct answer in the "Tutor Response" dialog:



Tutor mode limitations

Tutor mode can be enabled when stepping through any model, and it should work as described for most of the models in the ACT-R tutorial. However, it may have problems if productions use the !bind! and !mv-bind! operators, when there are values (buffer names, slot names, or slot contents) which have an "=" character in them, or there are variables used in the expression of a !eval! operator.

Step All

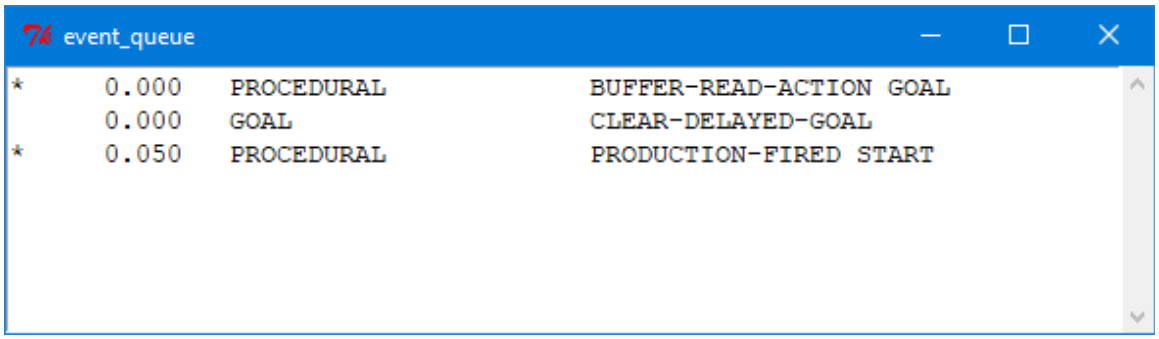
The "Step All" checkbox can be used to force the Stepper to stop on all events that are evaluated by ACT-R and not just those which are shown in the trace. That can be useful if one wants to pause or step through a model which does not have the trace enabled or one wants to step through events which are not shown in the trace either because they are below the current detail level or because they are internal events which are not displayed in the trace under any detail level.

Pause

The "Pause" button can be used to pause a running model. It will do so by setting the "Step All" flag and then opening the Stepper so that it will stop on the next available event. After pressing Pause, when the Stepper is closed it will turn the "Step All" flag off it was not on before Pause was pressed.

Event Queue

The "Event Queue" button is used to see all of the events currently scheduled in ACT-R. When it is pressed it will open the event queue window if it is not already open. If it is open, then pressing this button will bring it to the front – there can be only one event queue window open in the Environment even if there are multiple models since all of the models are executing events from the same queue. It shows the output from the ACT-R mp-show-queue command which displays all of the events that are currently scheduled.



Only the events marked with a "*" at the start of the line will be displayed in the model's trace with the current parameter settings, and thus be available to the Stepper if in use.

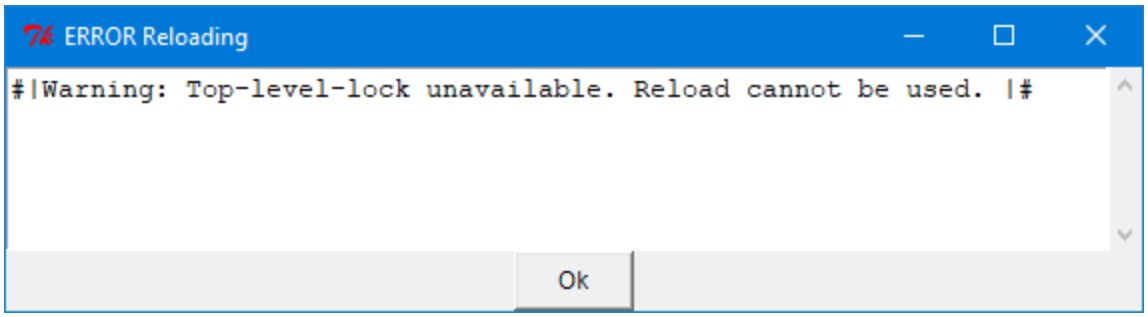
Reset

The "Reset" button is used to return models to their initial state. Pressing the "Reset" button is equivalent to calling the ACT-R **reset** command.

Reload

The "Reload" button is used to load the last model file which was loaded into ACT-R again (in this context, a model file is defined as any file which contains a call to the ACT-R **clear-all** command at the top-level). Pressing the "Reload" button is mostly equivalent to calling the ACT-R **reload** command. The "Reload" button will not function if ACT-R is currently running or if the Stepper tool is open, In those cases it will open a dialog with

this warning to indicate that:

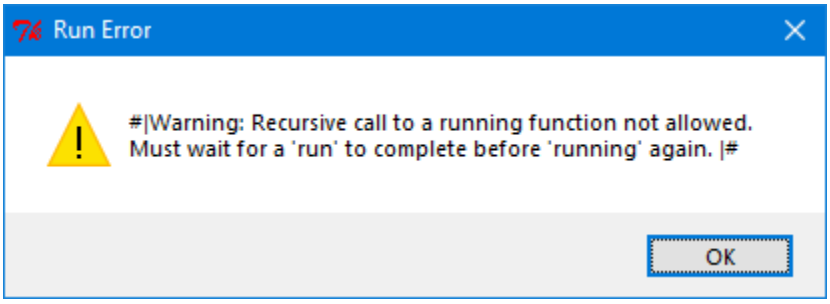


That dialog should be closed by pressing the "Ok" button before continuing with any of the other tools.

The Reload button differs slightly from calling the reload command because if the button is used to reload a model the Environment will not close the inspector windows (if that option is enabled) as a result of the clear-all call in the model file.

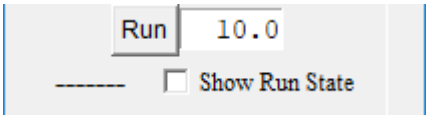
Run

The Run button can be used to run ACT-R. It is equivalent to calling the run command in ACT-R with the time indicated in the text box to the right of the Run button. The Run button cannot be used if ACT-R is already running. Pressing it when ACT-R is running will open a dialog window like this to indicate the problem:

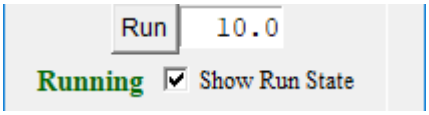


Running Indicator

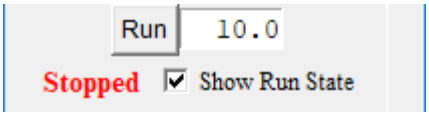
Below the Run button is a text item which can indicate whether or not ACT-R is currently running and a checkbox to enable that display. By default the running indicator is disabled – the box will not be checked and the text will show only dashes:



If you check the box then the text will update to show whether ACT-R is running or not regardless of how the run was initiated i.e. it does not require that the "Run" button be used to run the system to see the current state (note however that when first enabled it will not update until the next change in ACT-R's running state). If ACT-R is running it will say "Running" in green text:



If ACT-R is not running it will say "Stopped" in red text:



Enabling that display may cause a significant increase in the time required to run a model on some machines which is why it is disabled by default.

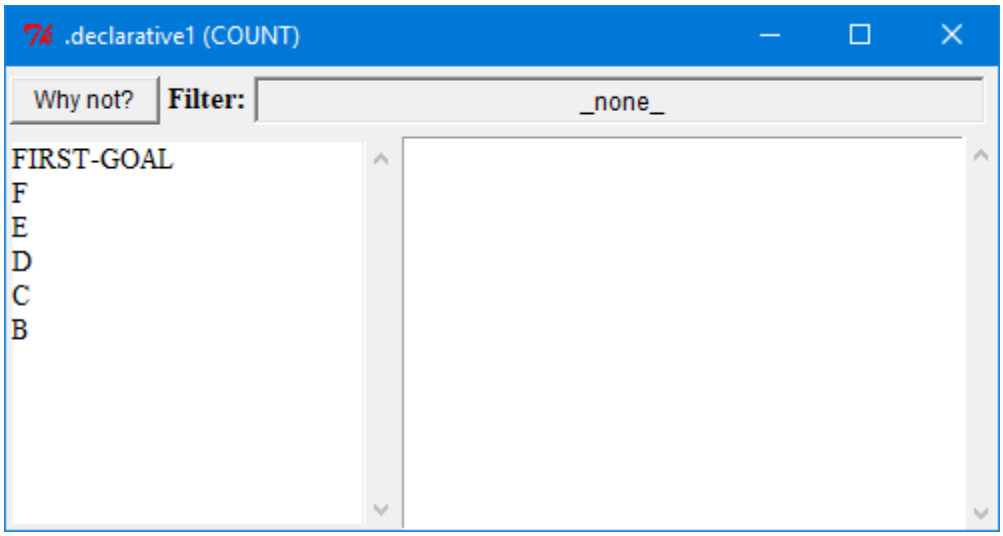
Current Data

The Current Data section of the Control Panel contains buttons for viewing detailed information about particular components of a model in its current state. These buttons can be useful in conjunction with the Stepper to view the details before and after an event occurs.

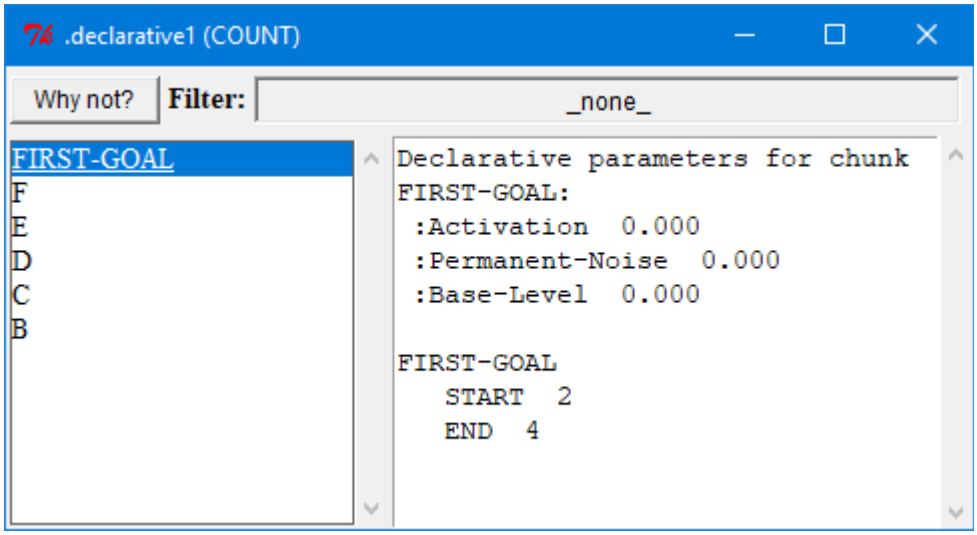
The contents of these windows are not updated automatically while ACT-R is running, but selecting a window will cause it to update the information which it contains. This differs from the ACT-R Environment in versions of ACT-R prior to 7.6 which updated everything continuously. With this version there is typically no noticeable performance penalty for having open inspection tools and it is also possible to compare the information from different times during a run (like the declarative parameters for a chunk) if multiple inspectors are used. There is however an option (which is enabled by default) that will force the windows to update automatically when the Stepper is being used to step through a run.

Declarative

The declarative tool allows the user to inspect the chunks in a model's declarative memory. Pressing the "Declarative" button opens a new declarative window for inspecting the declarative memory of the currently selected model and any number of such windows may be open at the same time. This is what a declarative viewer will look like (from the count model in unit 1 of the tutorial):



The list on the left shows all the chunks in declarative memory by default (see filter below for how to change that). Selecting one of those chunks will then cause the details of that chunk to be displayed in the window on the right:

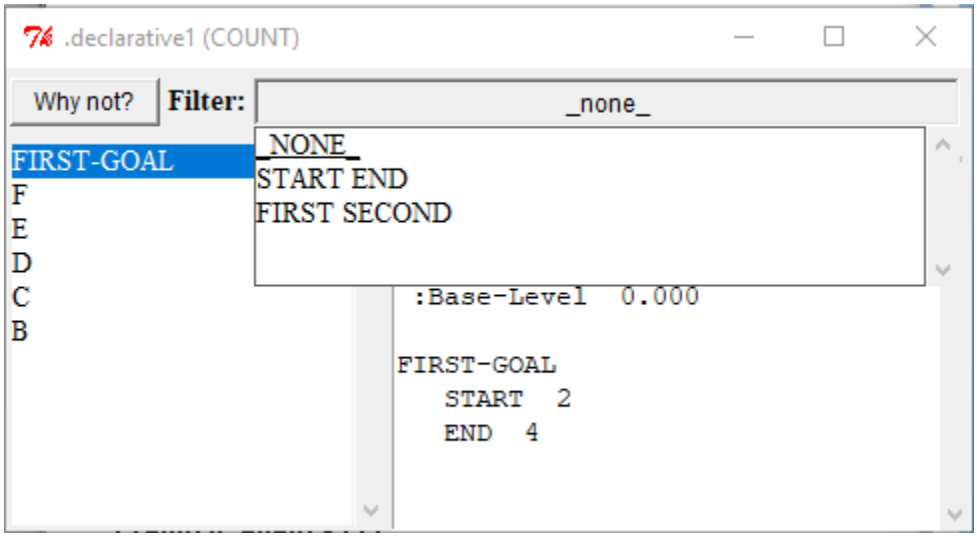


The chunk will be printed in the window, and if the subsymbolic computations are enabled then the window will also show the chunk's parameters as reported by **sdp** at the top.

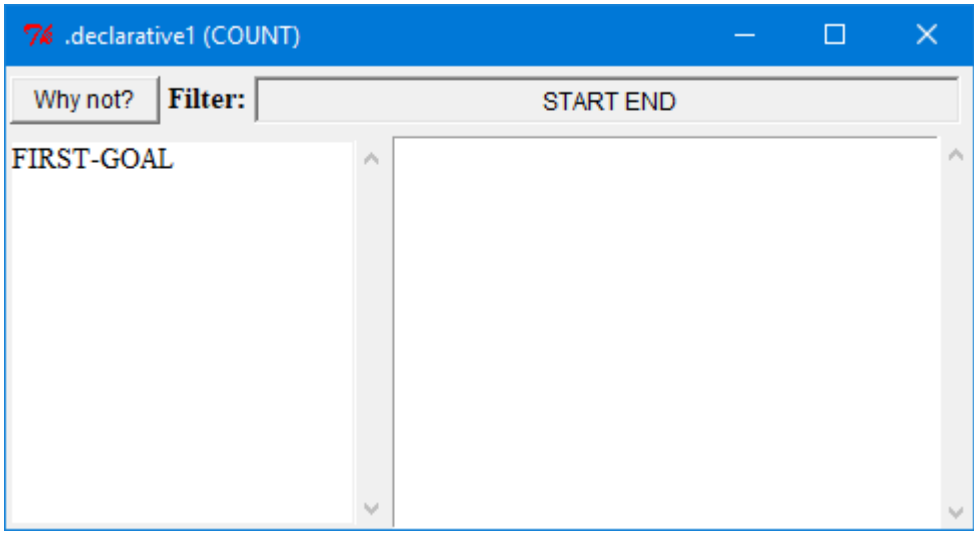
Filter

At the top of the window is a filter which allows one to restrict the display to only chunks

which have a particular set of slots. The default value of "_none_" means that all chunks in declarative memory will be displayed. To change the filter, click on the box containing the current filter value. That will open a window which contains lists of all the different sets of slots which exist among the chunks in declarative memory. If you pick a set of slots from that list that will be displayed in the filter and only chunks which contain that set of slots will be displayed. Here are the available sets of slots for the count model:



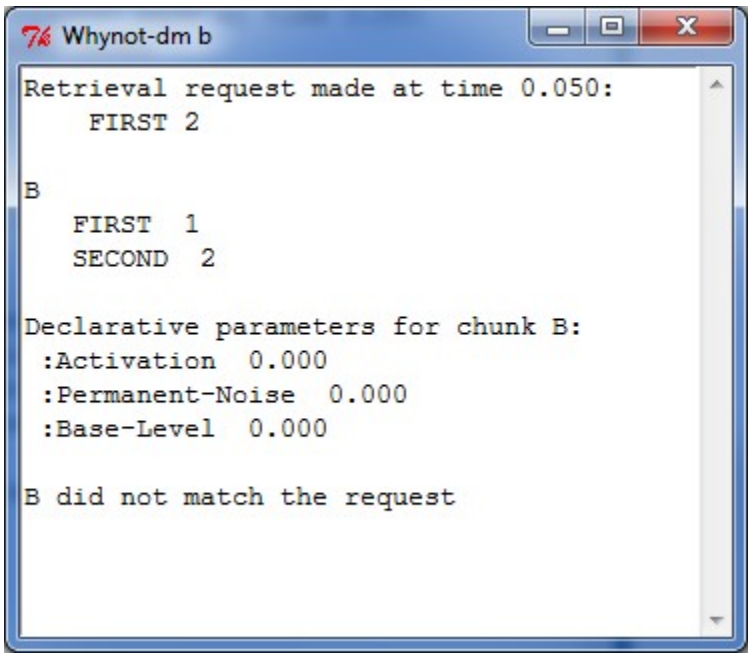
Here is the result of selecting the "start end" set of slots:



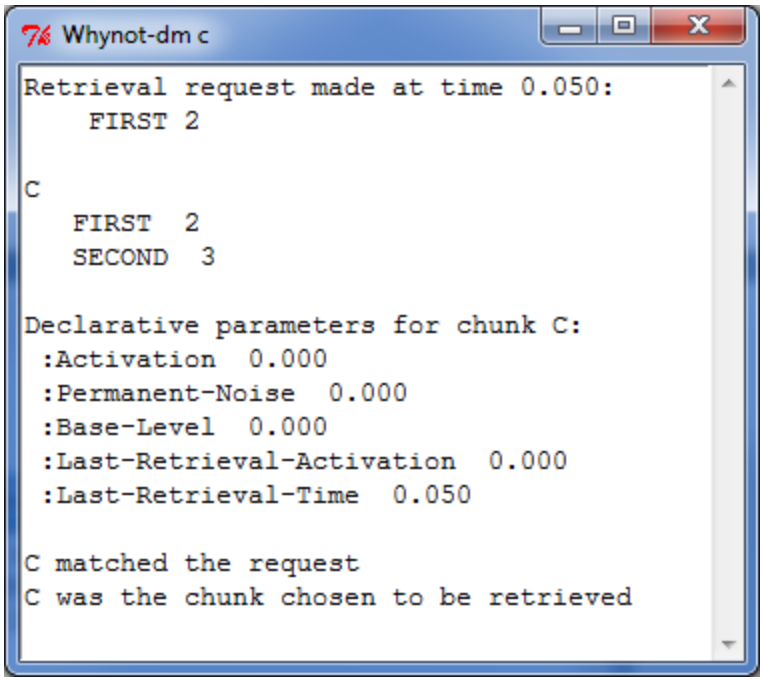
Why not?

The “Why not?” button at the top of the declarative window can be used to get information about whether a chunk was retrieved or not during the last retrieval request the model made. Pressing the “Why not?” button will open another window and display the results of calling the ACT-R **whynot-dm** command for the currently selected chunk in the declarative viewer.

The Whynot window will display the last retrieval request the declarative memory module received and then display the details of the selected chunk and indicate whether or not it matched that request. Here is a Whynot display for the chunk b .1 seconds into the run (after the model makes its first retrieval request):

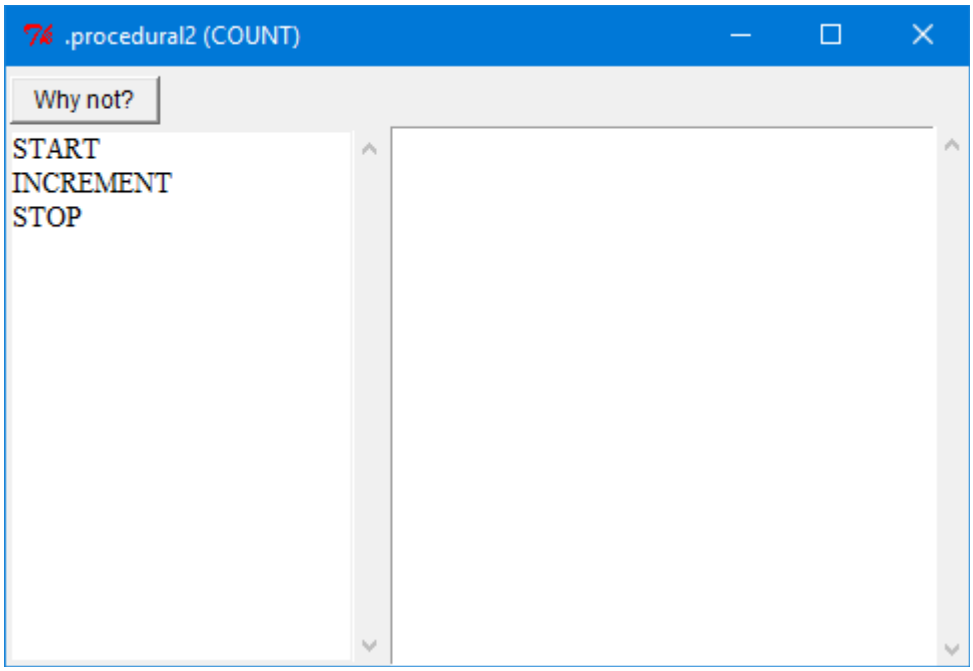


Here is a Whynot window showing result for the chunk c at that same time:

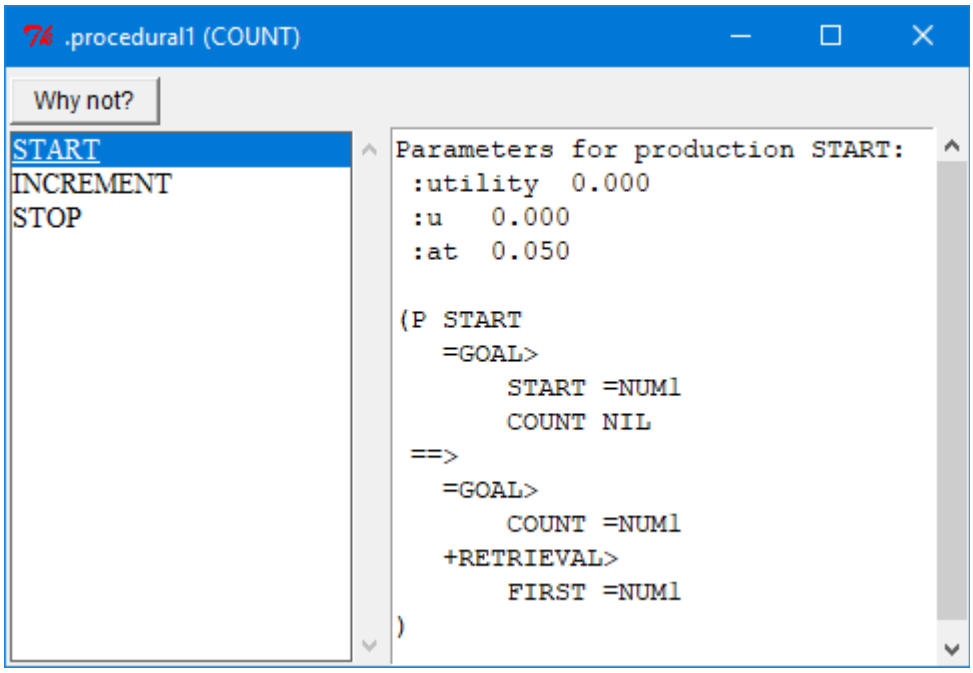


Procedural

The procedural tool allows the user to inspect the productions in the model's procedural memory. Pressing the "Procedural" button opens a new procedural window for inspecting the productions of the currently selected model and any number of such windows may be open at the same time. This is what a procedural viewer will look like (from the count model in unit 1 of the tutorial):



The list on the left contains all of the productions in the model. Selecting one of those productions will cause the details of that production to be displayed in the window on the right:



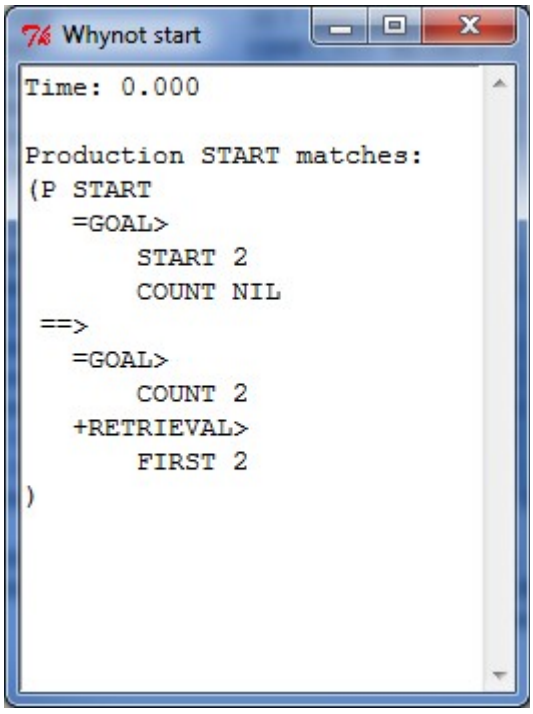
The production's text will be printed in the window, and if the subsymbolic computations are enabled then the production's parameters from **spp** are displayed at the top of the window.

Why not?

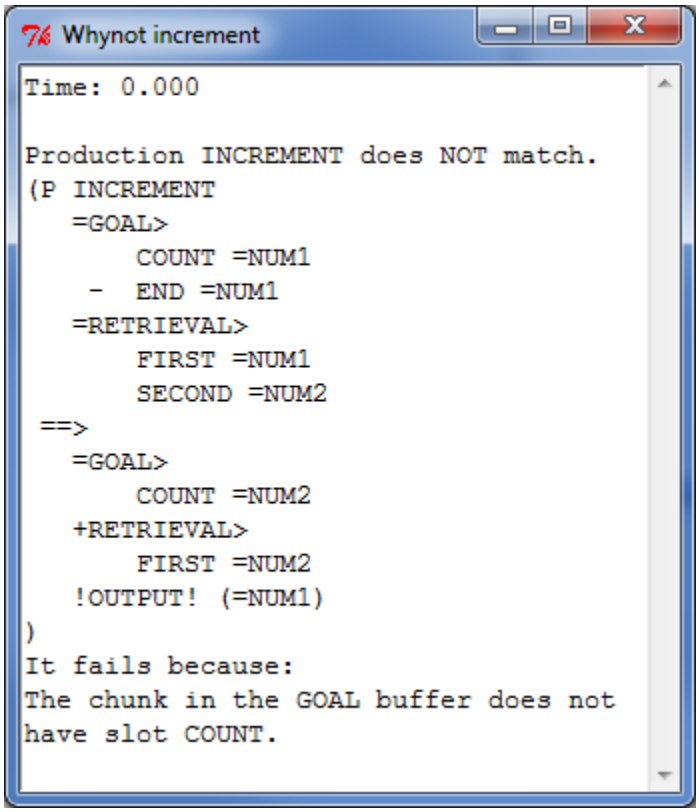
The "Why not?" button at the top of the procedural window is an important debugging aid. Pressing the "Why not?" button will open another window and display the results of calling the ACT-R **whynot** command for the currently selected production in the procedural window along with some other relevant information.

The Whynot window will display the time at which the whynot was generated and whether or not the LHS of that production currently matches. If it does match, that will be followed by the instantiation of the production. If it does not match, then it will print the text of the production and indicate the first condition which did not successfully match.

Here is a Whynot display for the start production in the count model at the beginning of the run when it matches:

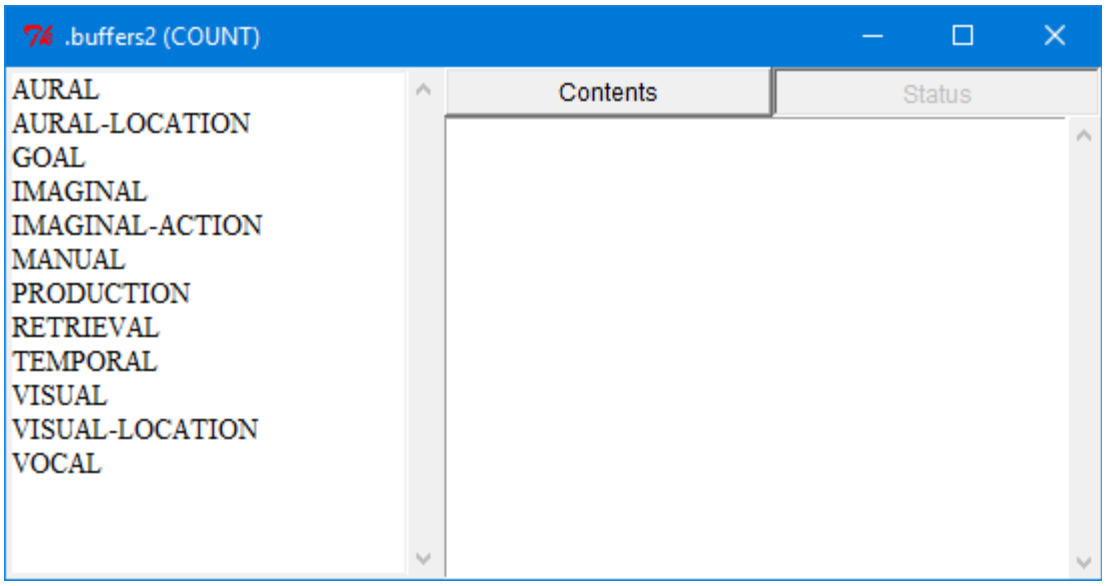


Here is a Whynot window showing the increment production at that same time which does not match:



Buffers

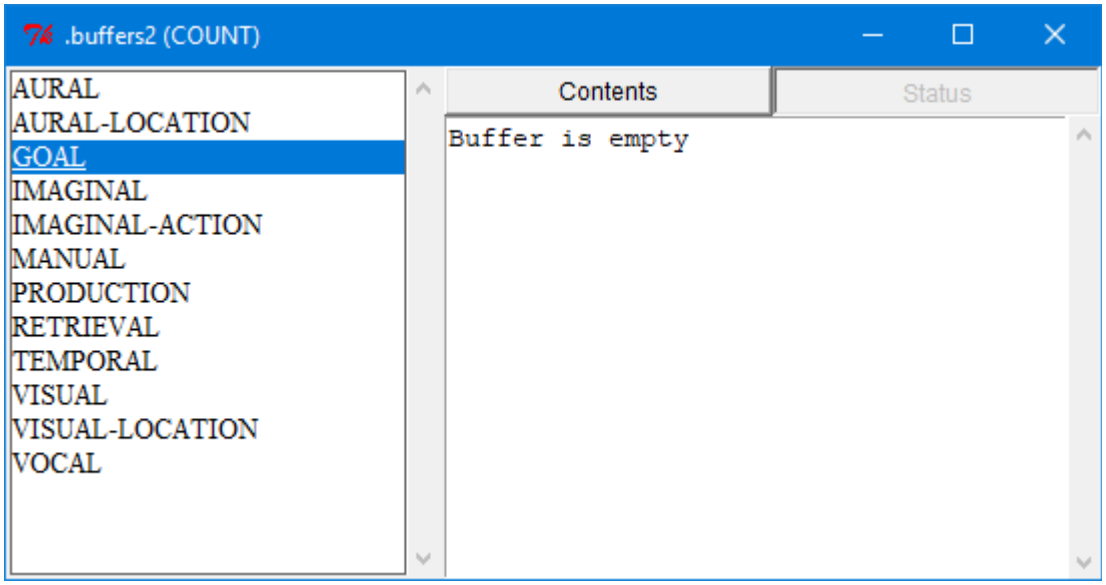
The buffer tool allows the user to inspect the chunks in the currently selected model's buffers and the current status of the model's buffers. Pressing the "Buffers" button opens a new buffer window for inspecting the buffers and any number of such windows may be open at the same time. This is what a buffer window will look like:



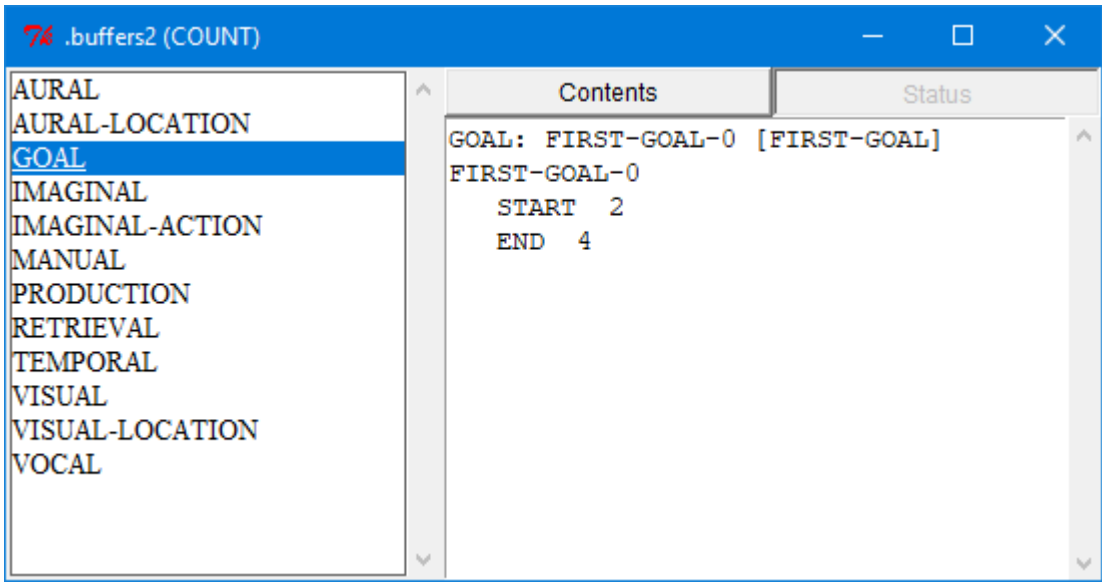
The list on the left shows the names of all the buffers in the model. Selecting a buffer from that list will show the information for that buffer in the window on the right. The two buttons at the top on the right indicate which information will be displayed.

Contents

If the "Contents" button is selected (the initial state when the window is opened) then it will show the chunk in the buffer. If the buffer is empty then it will print that:

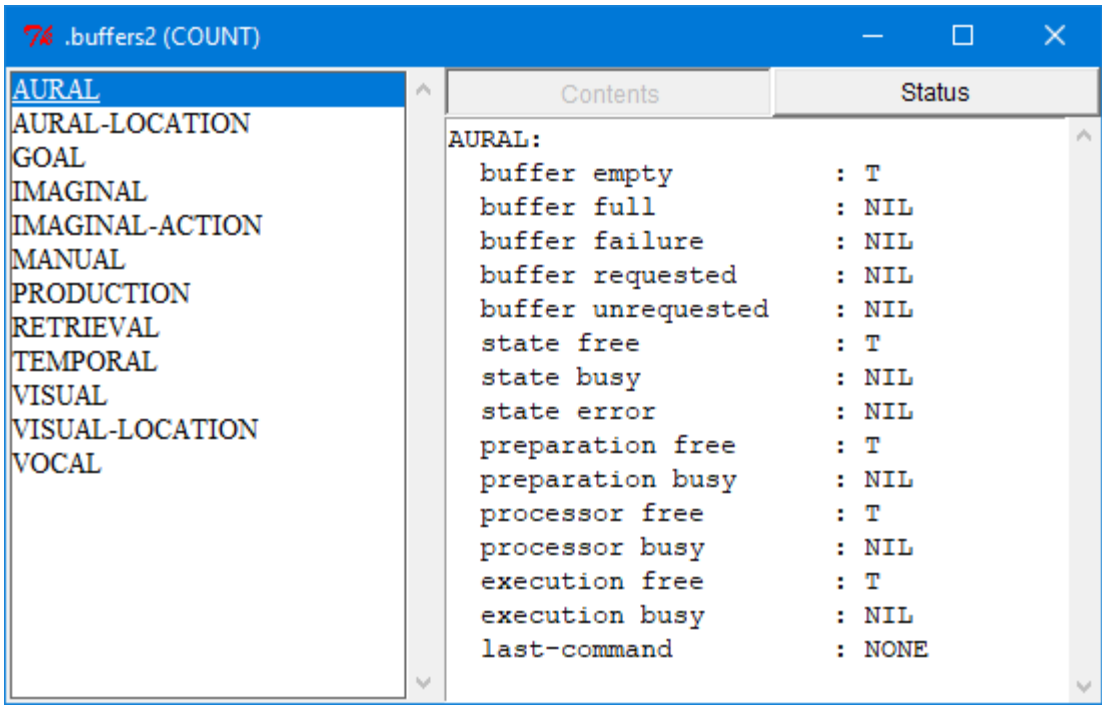


If there is a chunk in the buffer, then that chunk will be displayed using the **buffer-chunk** command (this is from the count model in unit 1 of the tutorial after it has started running):



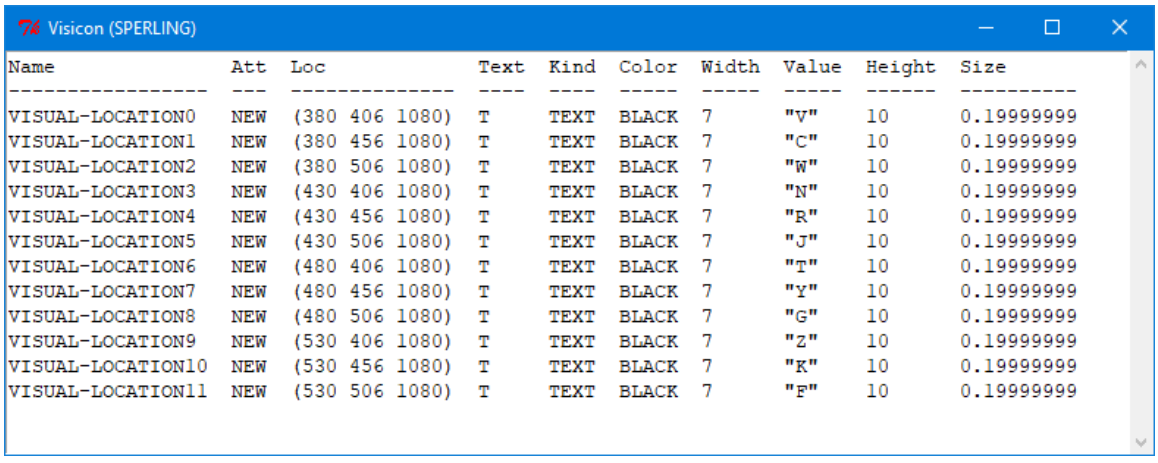
Status

If the “Status” button is selected then the status of the buffer is displayed using the **buffer-status** command for that buffer. The **buffer-status** command shows the queries available for the buffer along with whether or not that query is currently true (t) or false (nil), along with any additional status information provided by the module (as shown in the case above that there was no last-command received by the module from this buffer).



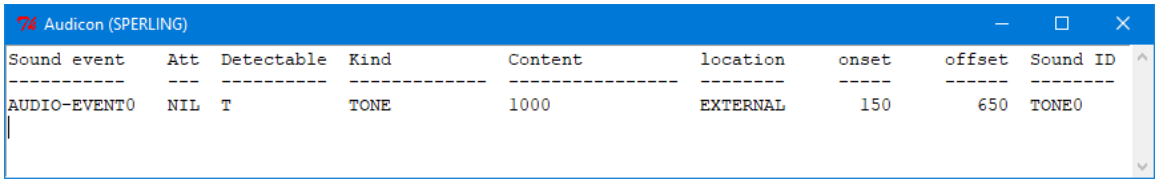
Visicon

Pressing the "Visicon" button will open a window showing the information currently available to the currently selected model's vision module. Only one such window will exist in the Environment for each available model. If a visicon window is already open for the current model, then pressing the button will bring that window to the front. The visicon window displays the information returned by the **print-visicon** command and here is an example using the sperling model from unit3 of the tutorial:



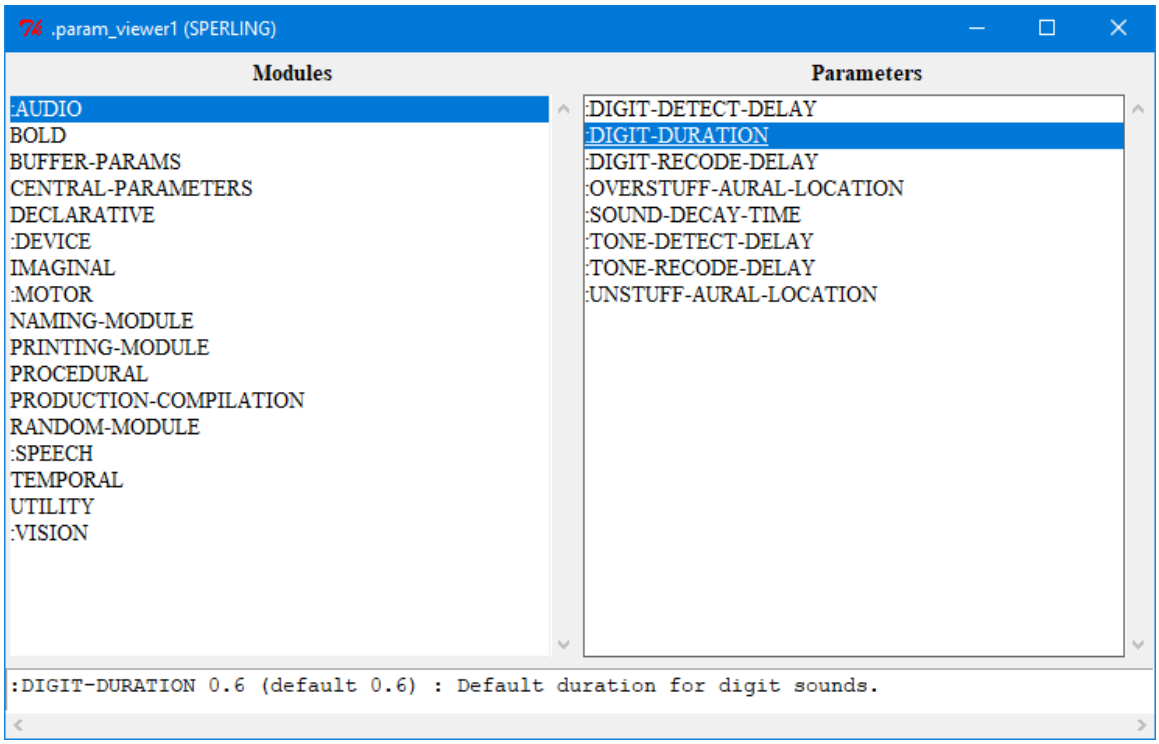
Audicon

Pressing the “Audicon” button will open a window showing the information currently available to the currently selected model’s audio module. Only one such window will exist in the Environment for each available model. If an audicon window is already open for the current model, then pressing the button will bring that window to the front. The audicon window displays the information returned by the **print-audicon** command and here is an example using the sperling model from unit3 of the tutorial:



Parameters

Pressing the “Parameters” button will open a window showing all of the currently defined modules and their parameters. Each time the button is pressed a new parameter viewer window will be opened for the current model. Selecting a module from the list will then display all of that module’s parameters in a list on the right, and selecting one of those parameters will display the current setting of that parameter, the default setting, and any documentation which the module provides for the parameter at the bottom of the window:


.param_viewer1 (SPERLING)
Modules
:AUDIO
BOLD
BUFFER-PARAMS
CENTRAL-PARAMETERS
DECLARATIVE
:DEVICE
IMAGINAL
:MOTOR
NAMING-MODULE
PRINTING-MODULE
PROCEDURAL
PRODUCTION-COMPILATION
RANDOM-MODULE
:SPEECH
TEMPORAL
UTILITY
:VISION
Parameters
:DIGIT-DETECT-DELAY
:DIGIT-DURATION
:DIGIT-RECODE-DELAY
:OVERSTUFF-AURAL-LOCATION
:SOUND-DECAY-TIME
:TONE-DETECT-DELAY
:TONE-RECODE-DELAY
:UNSTUFF-AURAL-LOCATION
:DIGIT-DURATION 0.6 (default 0.6) : Default duration for digit sounds.

General Recordable Data

The General Recordable Data tools provide a way for the user to have ACT-R record detailed information as the model runs and then be able to inspect that data after the model stops, using tools that allow one to view the data at particular times during the run. They also provide a way to save that recorded data to a file so that it can be loaded and inspected at a later time if desired (even without the model itself being available).

Opening one of these windows will make the model record the information requested, and that will be recorded as long as the window is open. Recording the information as a model runs requires additional resources. The model will take longer to run the more types of information that are being recorded, and there will be additional computer memory required to store that information. Therefore one probably doesn't want to always record everything, especially once the model is working and it's being run repeatedly to generate results. As an example, the zbrodoff model from unit 4 of the tutorial uses most of the modules in a model and running one block of the task covers about 540 seconds of model time. It takes about 2 seconds to run one block on my computer without recording any information. With only the text trace being recorded it takes about 8 seconds to run the model through one block, and the saved data file of that trace is around 2.8MB in size. With all of the recording tools open it takes about 22 seconds to run the model through one block.

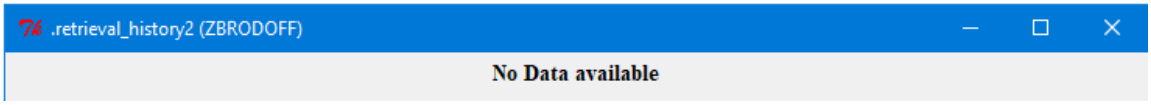
Common Functionality

All of the tools in this section have some capabilities in common which will be described first. This section will cover those commonalities and then the following sections will provide the specific details for each tool. There are four things which all of these tools share: a status line which shows information about the data being displayed and three buttons labeled "Get History", "Save History", and "Load History".

Status Line

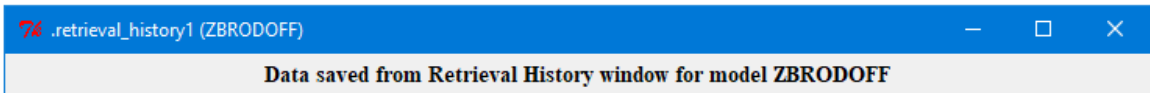
Usually at the top of the window (but not always) will be an area that will show the details

of the current data being displayed. If there was no data to display it will indicate that like this:



If the data being displayed was available from a model it will show something like this:

If the data being displayed came from a saved history file then it will show the comments from that history record, and the data saved from the Environment will always have this form:



Get History

The "Get History" button will get the current history information from the model and display it in the viewer.

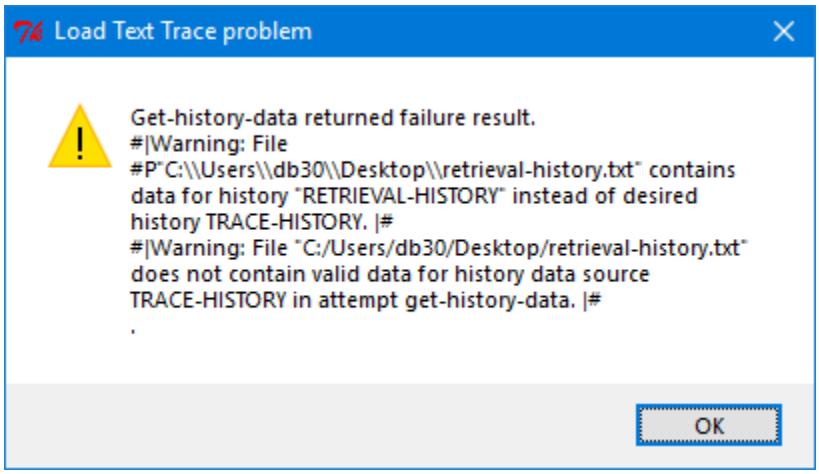
Save History

The "Save History" button will open a file creation dialog to provide a name for a file in which to save the history data. It will write the currently available data to that file (whether or not it is being currently displayed).

Load History

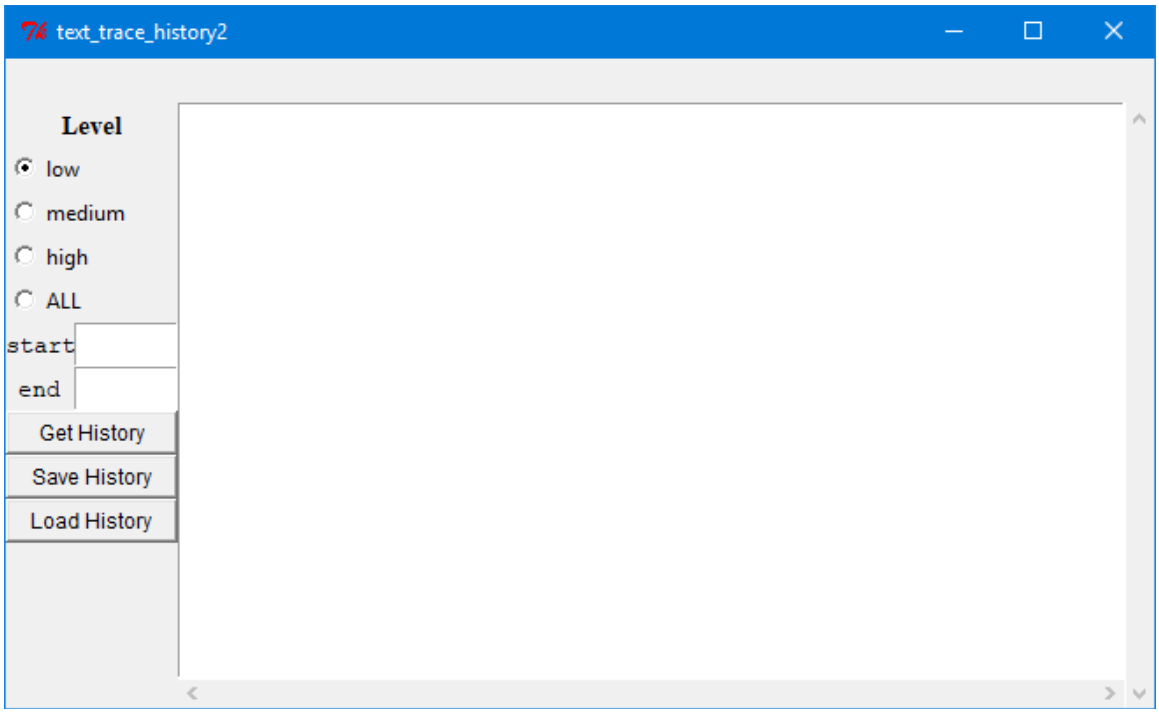
The "Load History" button will open a file selection dialog to choose a file which contains saved history data which is appropriate for the current history viewer. The history data

will be read from that file and displayed if it is valid. If it is not valid a dialog will be shown to indicate the problem. Here is an example where a file containing the wrong data was chosen:



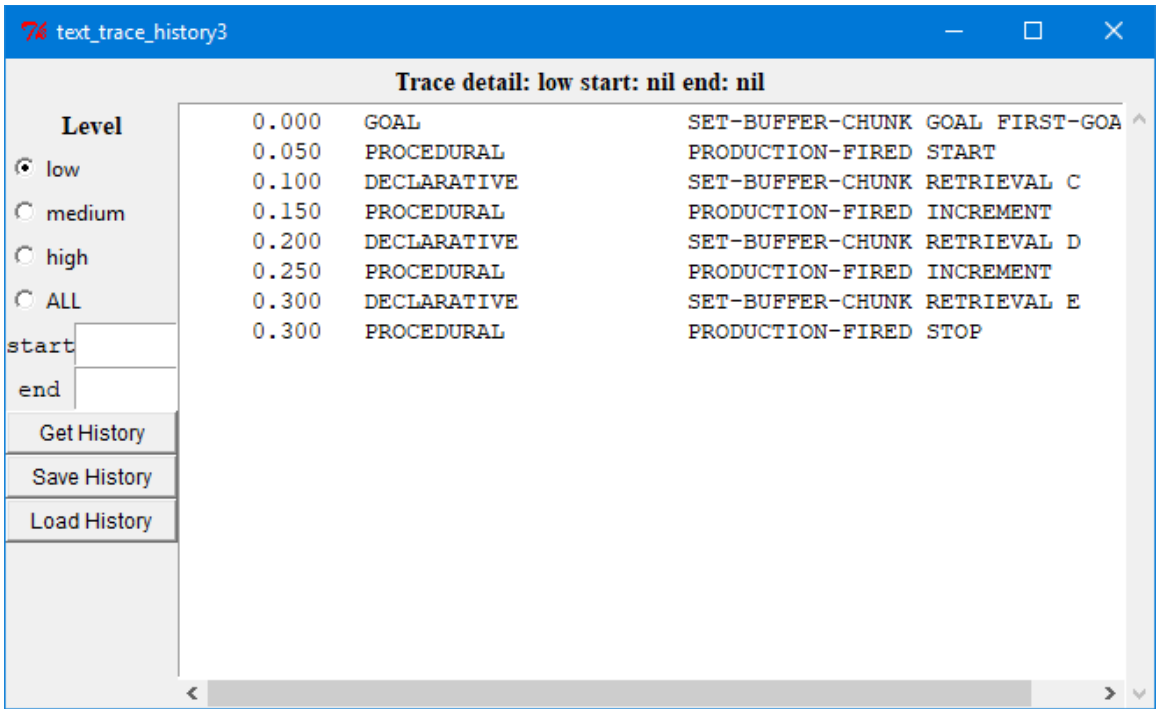
Text Trace

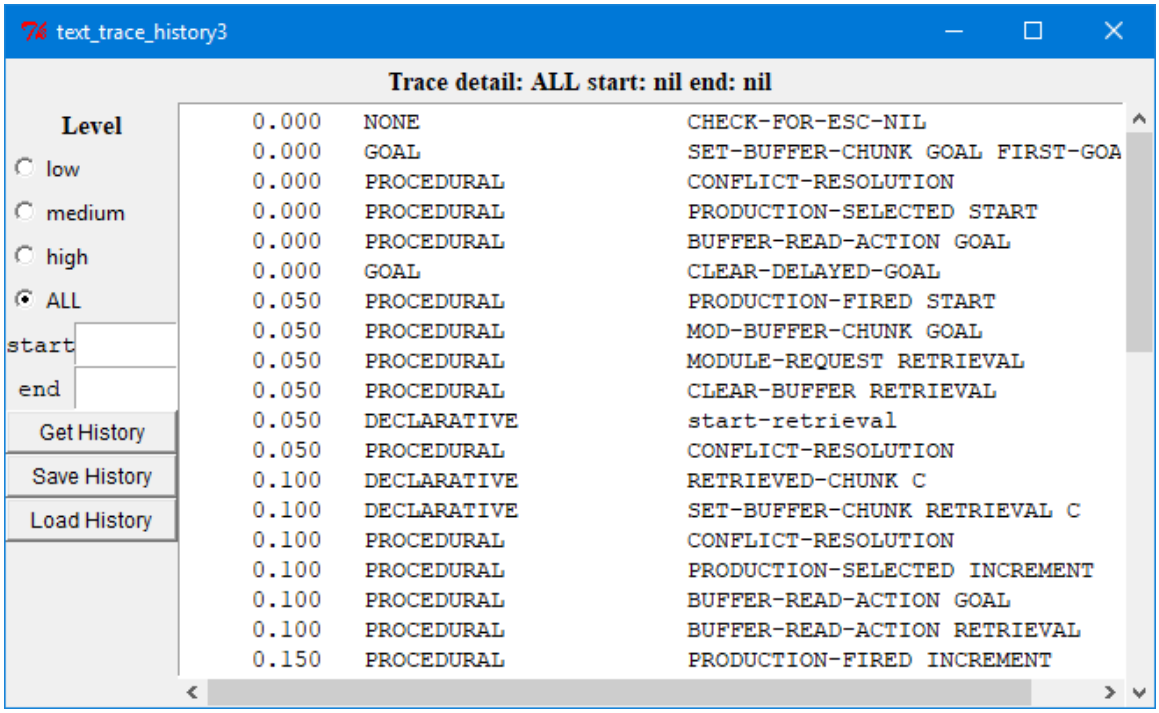
This button opens a new “Text Trace” window for the current data source and any number of those windows may be open. The tool allows one to see the event trace generated by the model as it ran even if that trace was not being displayed during the run. Here is the initial display of the window without any data shown:

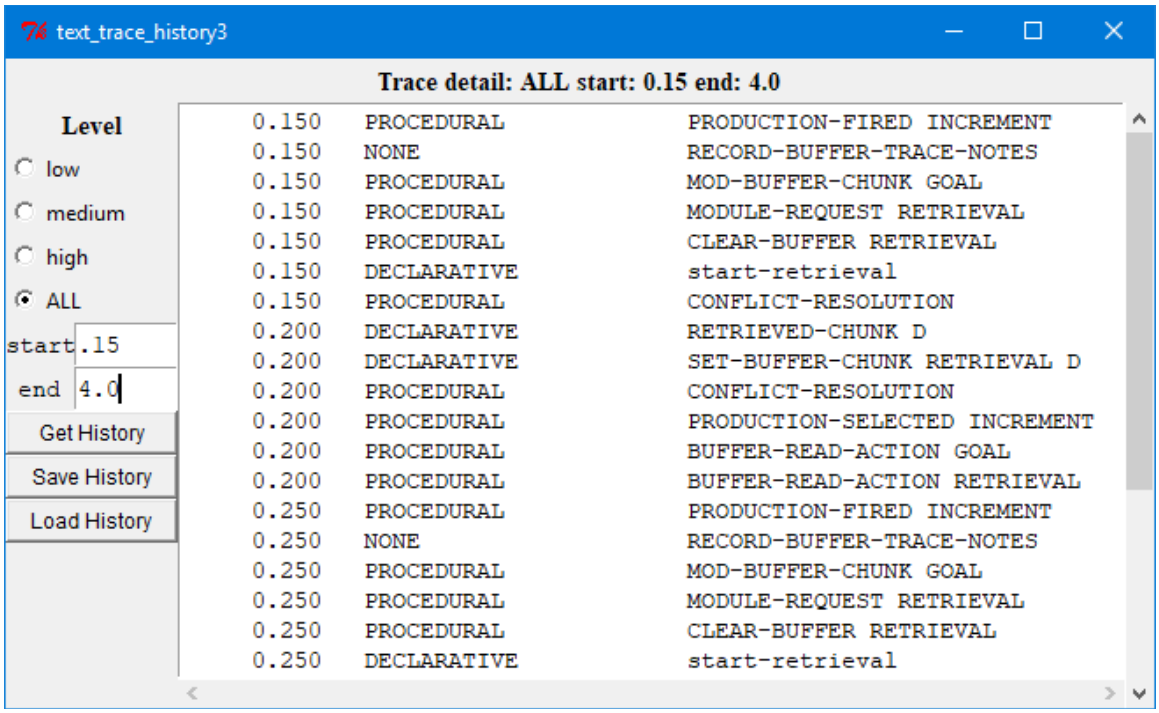


When you get the data it will be displayed in the window, and how much data is displayed is based on the settings selected on the Left of the window (you must press "Get History" again to update the display if you change the settings). The radio buttons choose how much detail to display from the trace. The first three options, low, medium, and high, correspond to the setting of the **:trace-detail** parameter which can be set in a model. The last option, ALL, provides additional information beyond the high detail trace. ALL will display every event which occurred, including those which were marked as having no output (which are typically internal system maintenance actions). You can also use the start and end settings to restrict the trace to a particular segment of time specified in seconds. If no values are provided for start and end they will be time 0 and current model time respectively.

Here are some examples using the count model from unit 1 of the tutorial:

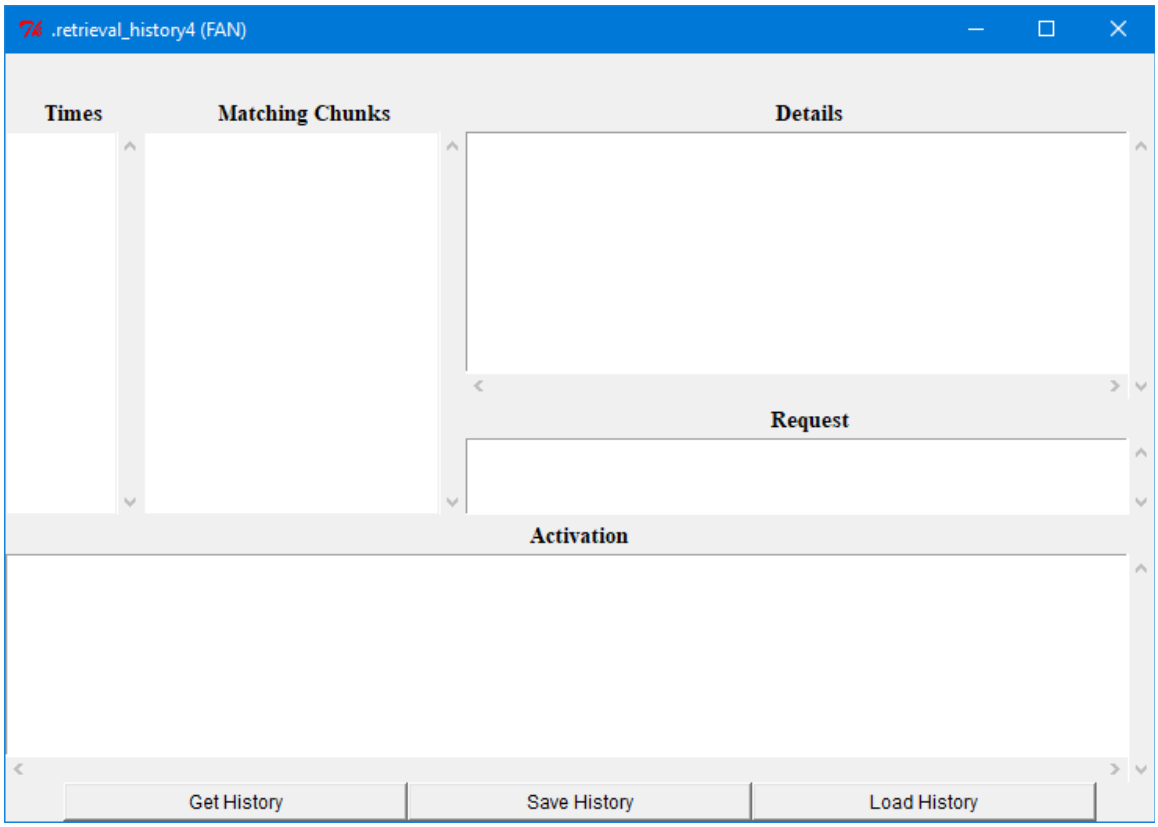

text_trace_history3
Trace detail: low start: nil end: nil
Level
low
medium
high
ALL
start
end
Get History
Save History
Load History
0.000 GOAL SET-BUFFER-CHUNK GOAL FIRST-GOA
0.050 PROCEDURAL PRODUCTION-FIRED START
0.100 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL C
0.150 PROCEDURAL PRODUCTION-FIRED INCREMENT
0.200 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL D
0.250 PROCEDURAL PRODUCTION-FIRED INCREMENT
0.300 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL E
0.300 PROCEDURAL PRODUCTION-FIRED STOP


text_trace_history3
Trace detail: ALL start: nil end: nil
Level
low
medium
high
ALL
start
end
Get History
Save History
Load History
0.000 NONE CHECK-FOR-ESC-NIL
0.000 GOAL SET-BUFFER-CHUNK GOAL FIRST-GOA
0.000 PROCEDURAL CONFLICT-RESOLUTION
0.000 PROCEDURAL PRODUCTION-SELECTED START
0.000 PROCEDURAL BUFFER-READ-ACTION GOAL
0.000 GOAL CLEAR-DELAYED-GOAL
0.050 PROCEDURAL PRODUCTION-FIRED START
0.050 PROCEDURAL MOD-BUFFER-CHUNK GOAL
0.050 PROCEDURAL MODULE-REQUEST RETRIEVAL
0.050 PROCEDURAL CLEAR-BUFFER RETRIEVAL
0.050 DECLARATIVE start-retrieval
0.050 PROCEDURAL CONFLICT-RESOLUTION
0.100 DECLARATIVE RETRIEVED-CHUNK C
0.100 DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL C
0.100 PROCEDURAL CONFLICT-RESOLUTION
0.100 PROCEDURAL PRODUCTION-SELECTED INCREMENT
0.100 PROCEDURAL BUFFER-READ-ACTION GOAL
0.100 PROCEDURAL BUFFER-READ-ACTION RETRIEVAL
0.150 PROCEDURAL PRODUCTION-FIRED INCREMENT

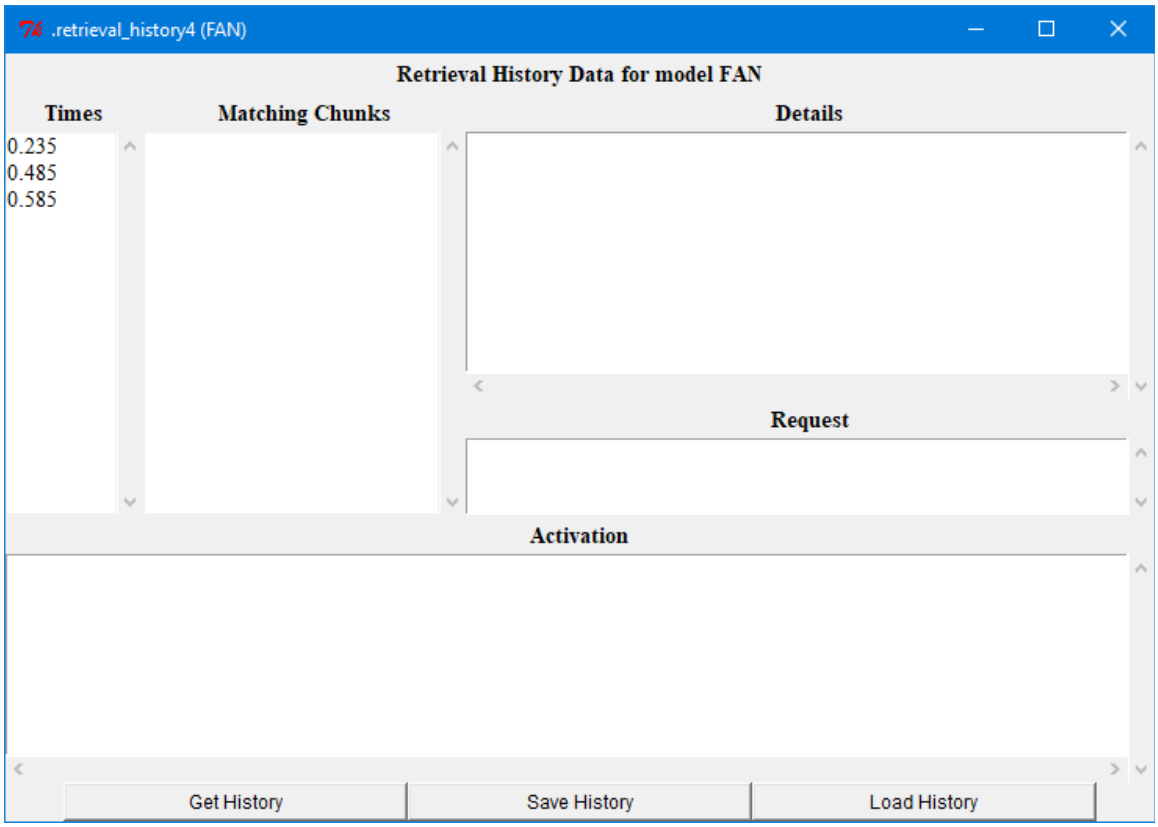


Retrieval History

This button opens a new “Retrieval History” window and any number of those windows may be open. It can be used to display information about all of the retrieval requests which have been made to the declarative module. Here is a display of the window without any data shown (how it will always appear upon opening):

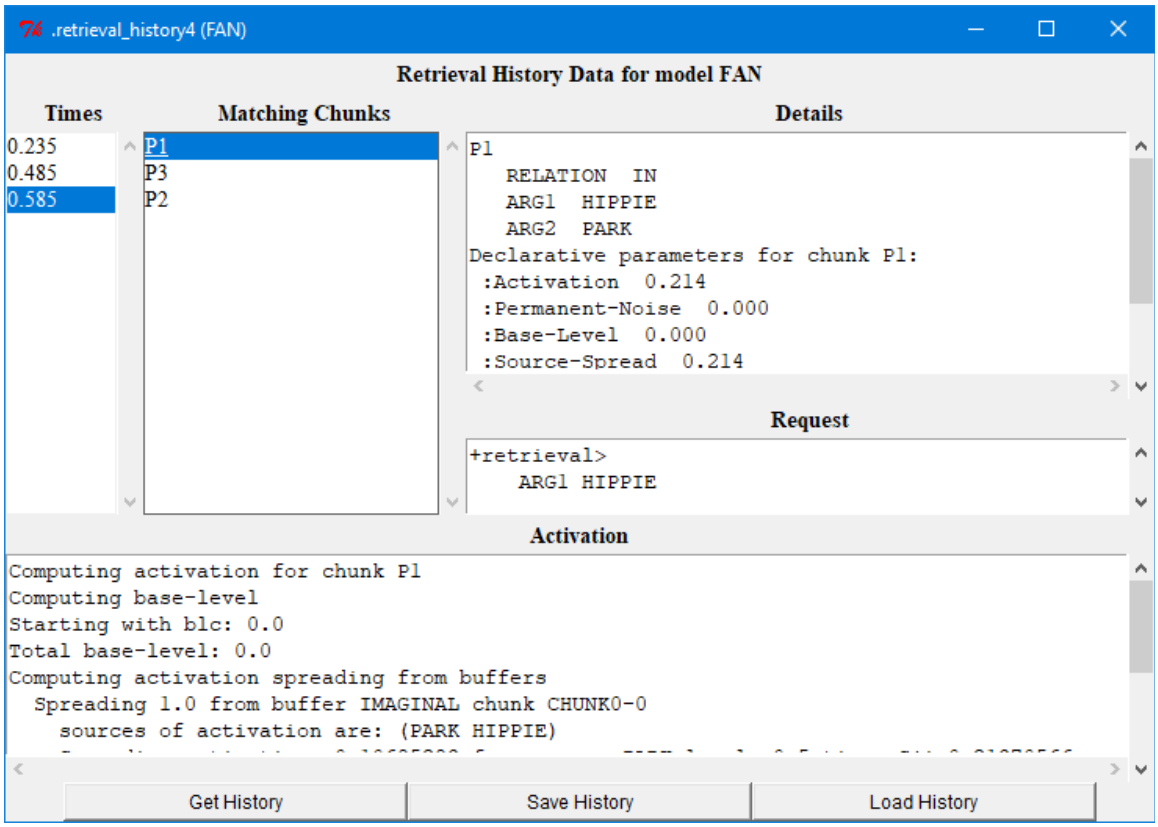


Here is the data after a run of the fan model from unit 5 of the tutorial for the sentence “the hippie is in the park” using the person as the retrieval cue:



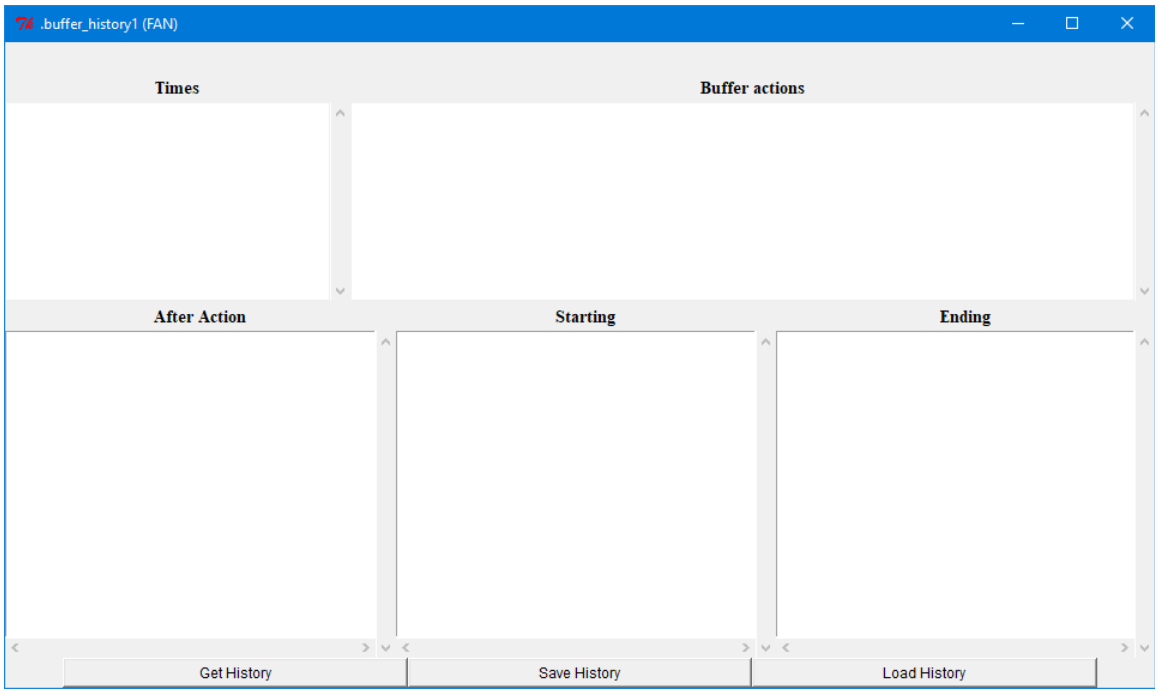
The left column displays all the times at which a retrieval request was made. Selecting one of those times will cause the "Matching Chunks" section of the window to list all of the chunks that were in declarative memory and matched the request at that time. The item at the top of the list is the one which was retrieved, or will be the keyword :retrieval-failure if no chunk was retrieved. The rest of the chunks in the list are in no particular order. The "Request" section of the window will display the request which was made at that time.

Selecting one of the chunks from the "Matching Chunks" list will result in the "Details" section being filed with a printing of the chunk along with the parameter values for that chunk at the time of the retrieval request. The "Activation" section of the display will show the detailed activation trace of how that chunk's activation was computed at that time. Here is the tool after selecting the 0.585 second time and the first chunk on the resulting list, p1:

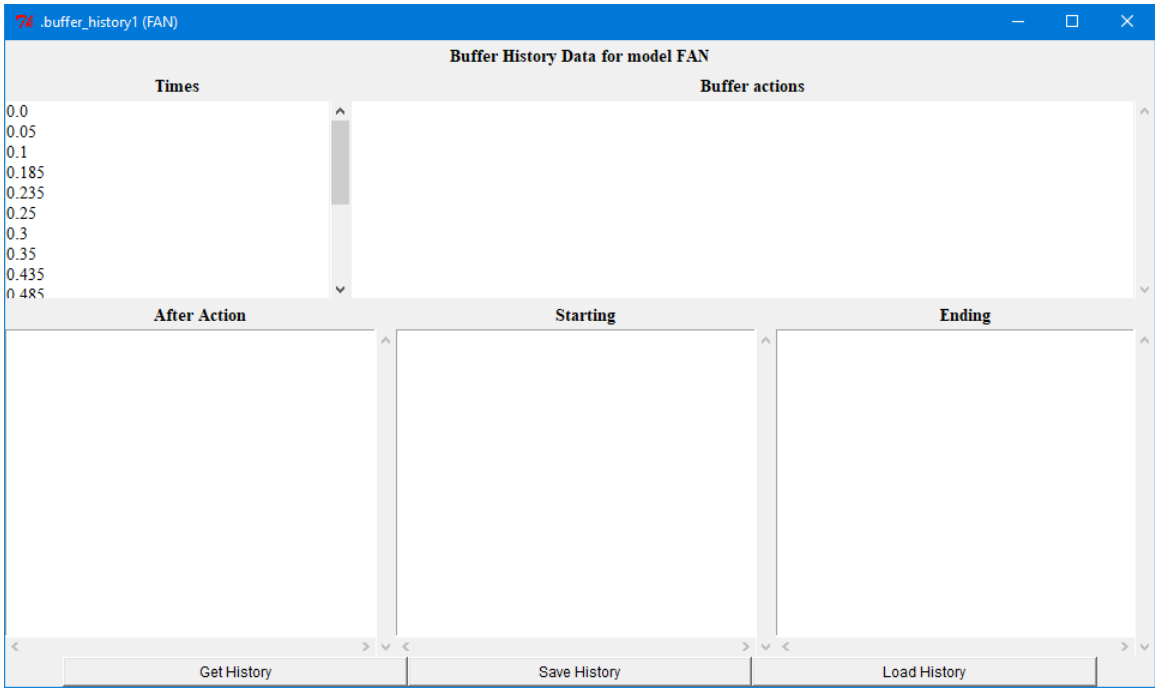


Buffer History

This button opens a new “Buffer History” window and any number of those windows may be open. Here is a display of the window without any data shown (how it will always appear upon opening):

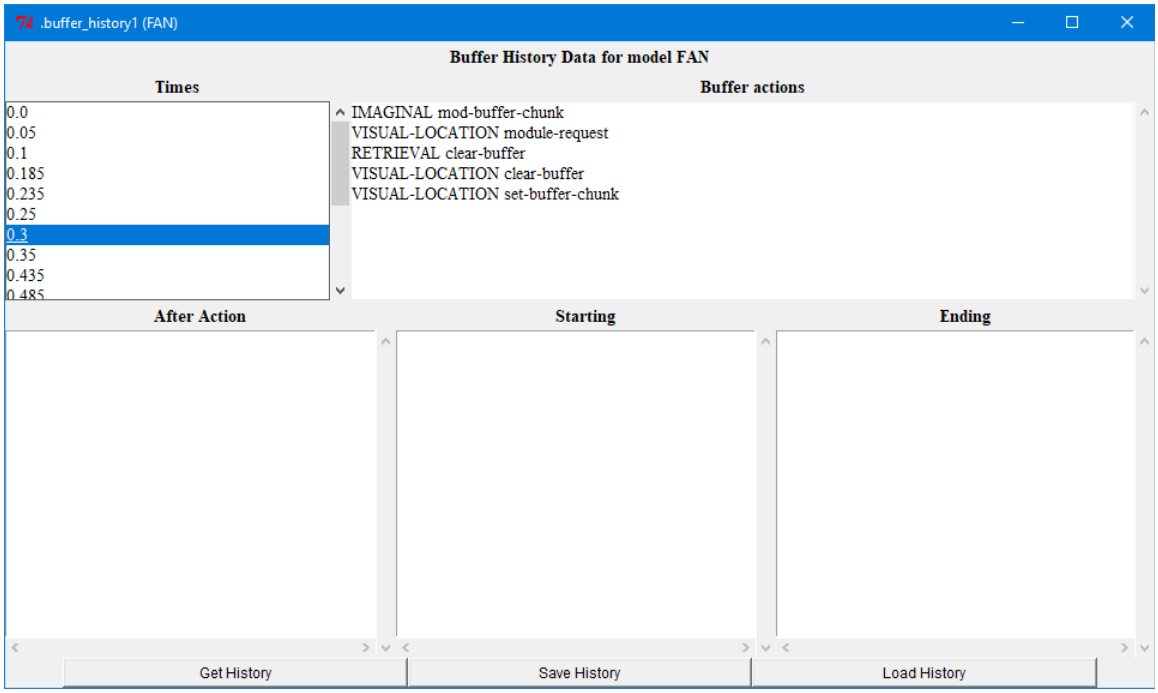


Here is the initial data after a run of the fan model from unit 5 of the tutorial for the sentence “the hippie is in the park”:

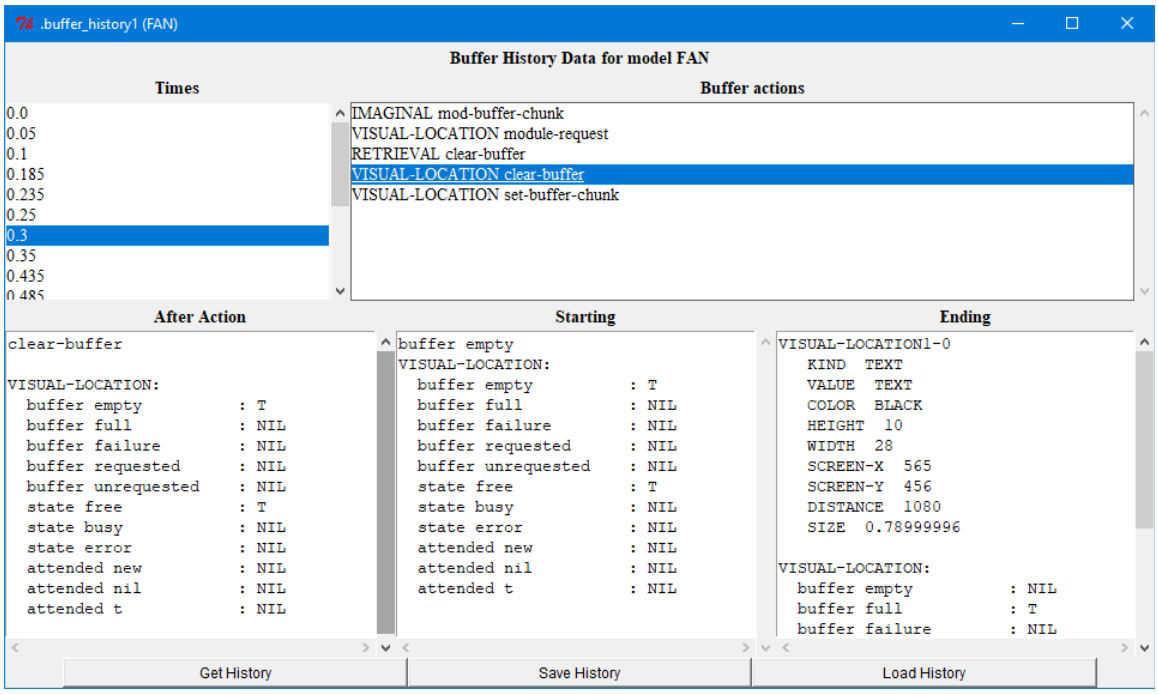


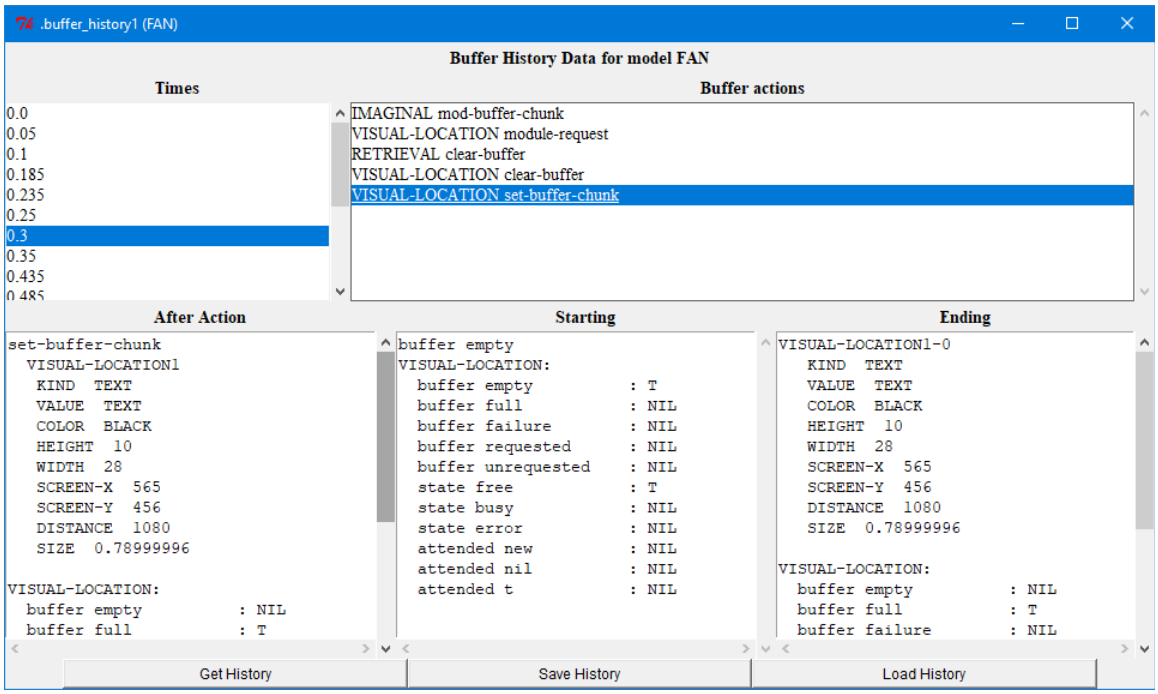
The left column displays all the times at which a change occurred in some buffer, where a

change is any of: the buffer clearing, a chunk being placed into the buffer, the chunk in the buffer being modified, or a request being made through the buffer. Selecting a time from that list will update the window to the right of the times to show the actions which occurred to buffers at that time. Here is the result of selecting the .3 time for the fan task:



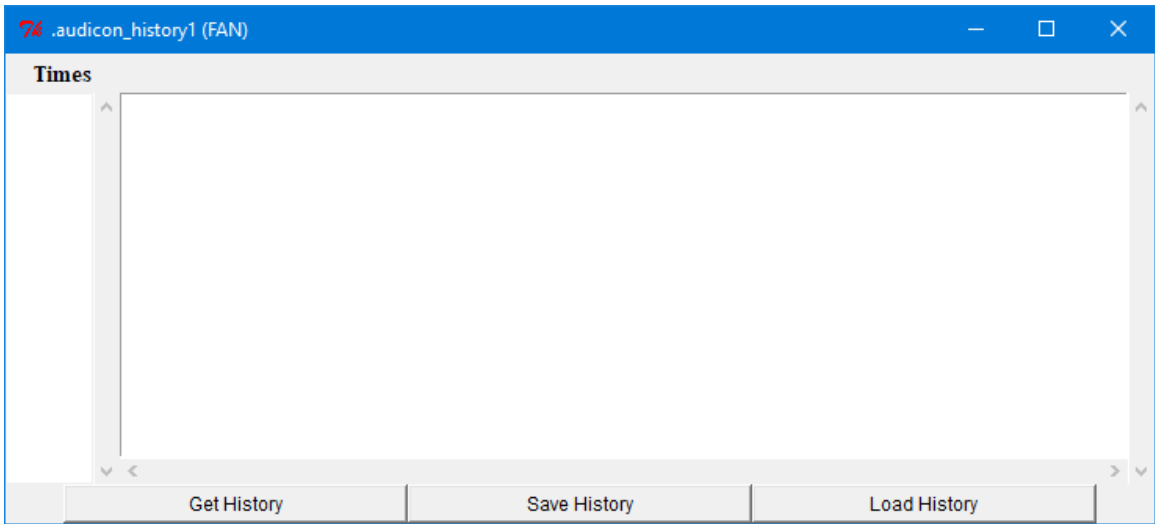
Selecting one of those actions will update the three windows at the bottom. The Starting and Ending windows will show the contents of that buffer (if it has a chunk) and the buffer status information for the buffer at the beginning of the actions which occurred at that time and after all of the actions which occurred at that time respectively. The After Action window will show the selected action which occurred, the contents of the buffer after that action (if it has a chunk), and the buffer status. Here are the displays when picking the second and third actions which occurred for the visual-location buffer at that time:



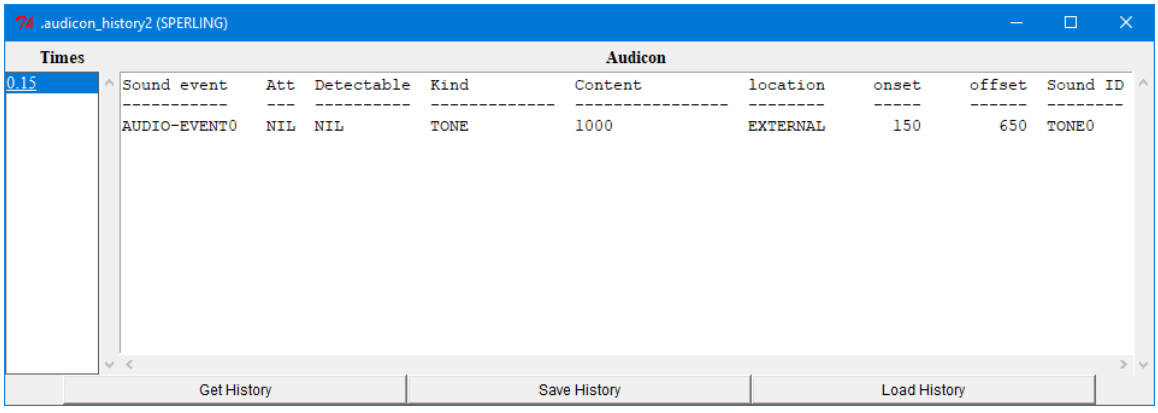


Audicon History

This button opens a new “Audicon History” window for and any number of those windows may be open. Here is a display of the window without any data shown (how it will always appear upon opening):

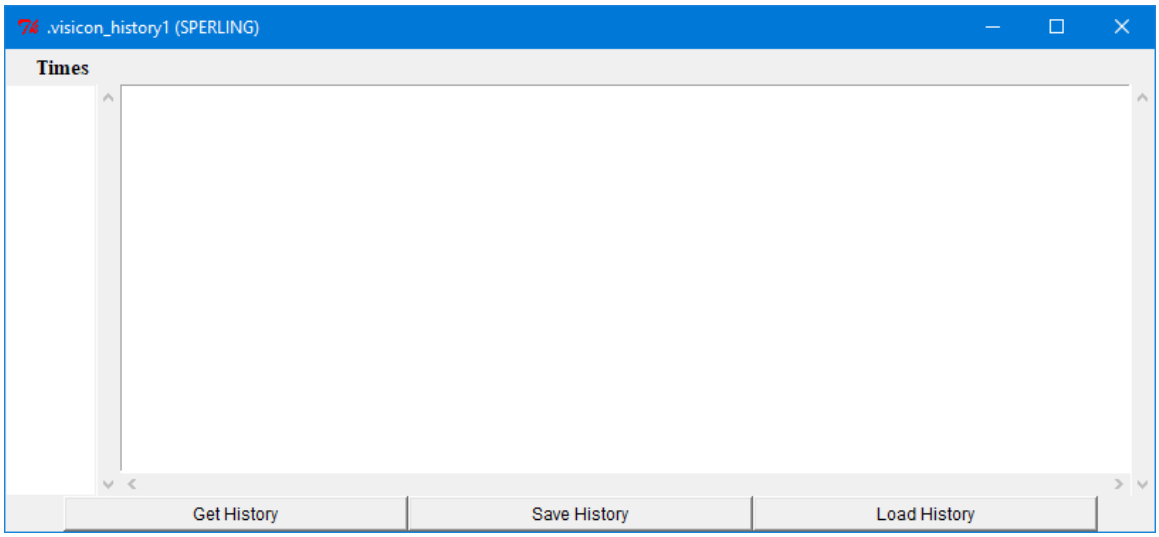


The audicon history monitors the new-sound action of the audio module and records the audicon information each time there is a new sound. This tool displays all the times when new-sound occurred and selecting one shows the audicon at that time. Here is the output from the sperling model in unit 3 of the tutorial after a trial at which the tone occurred .15 seconds into the task:

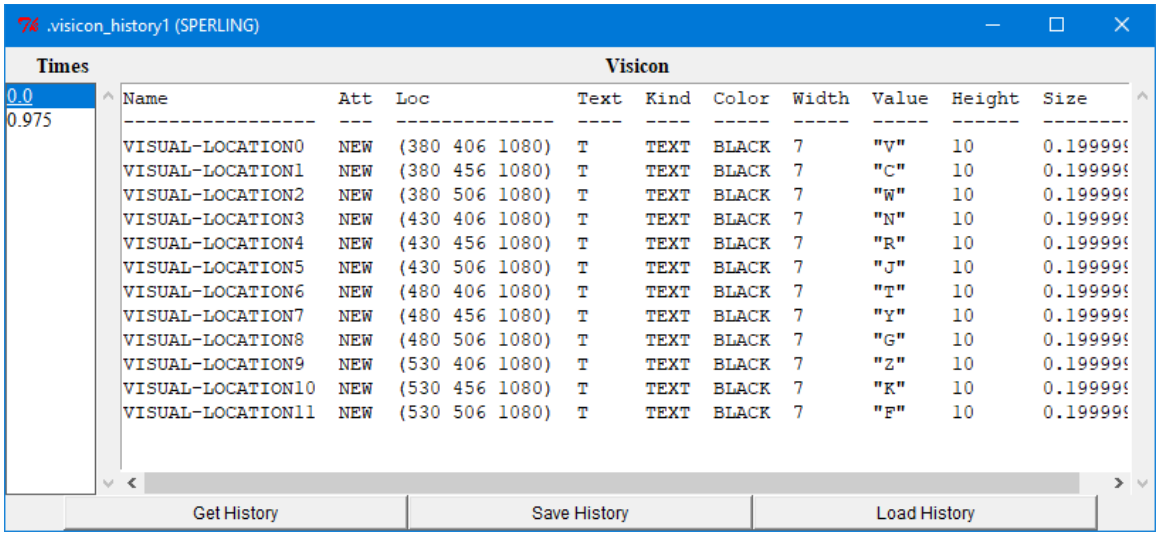


Visicon History

This button opens a new “Visicon History” window and any number of those windows may be open. Here is a display of the window without any data shown (how it will always appear upon opening):



The visicon history monitors the visicon-update action of the vision module and records the information each time that results in a change to the visicon. This tool displays all the times when such a change occurred and selecting one shows the visicon at that time. Here is the output from the sperling model in unit 3 of the tutorial after a trial at which the tone occurred .15 seconds into the task:

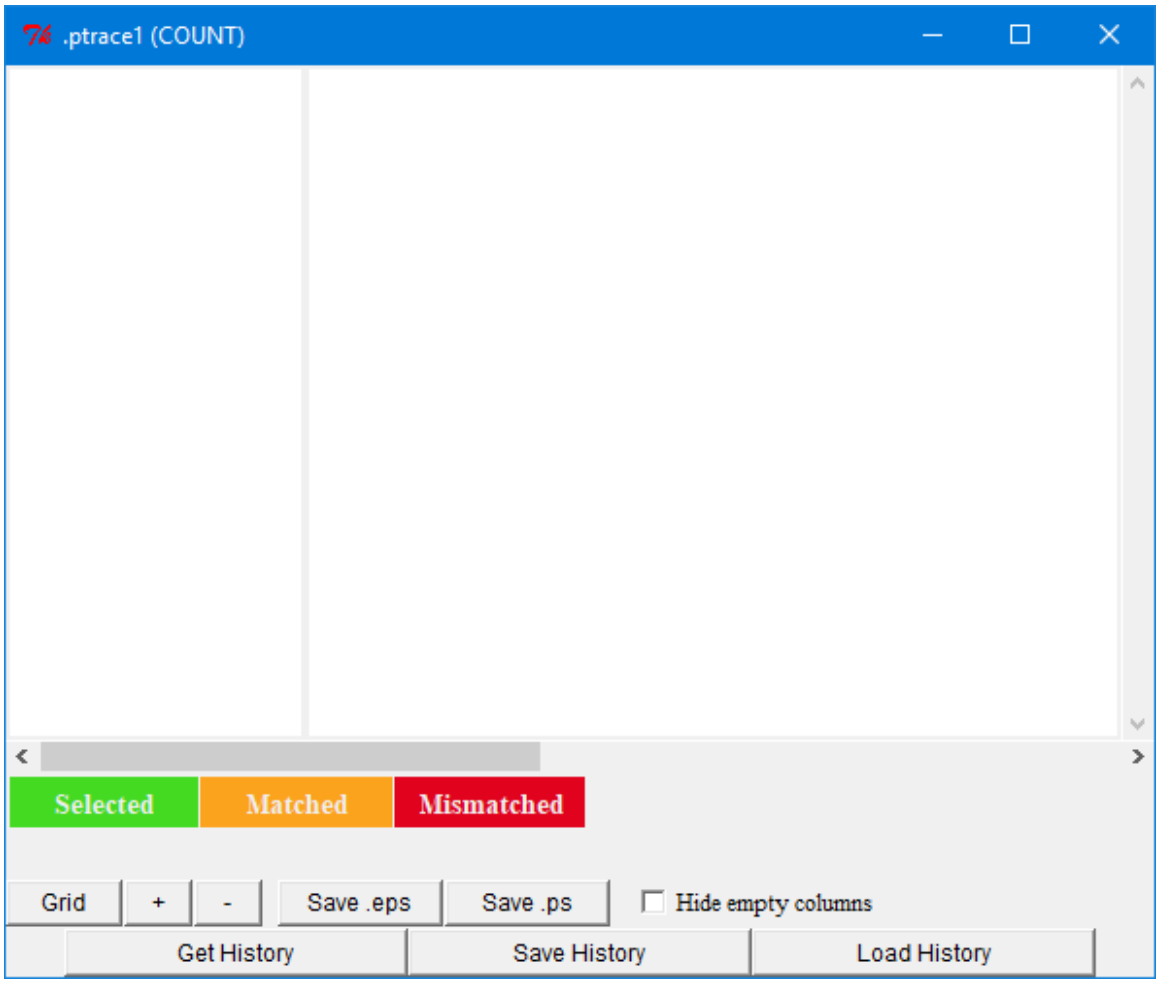


Production

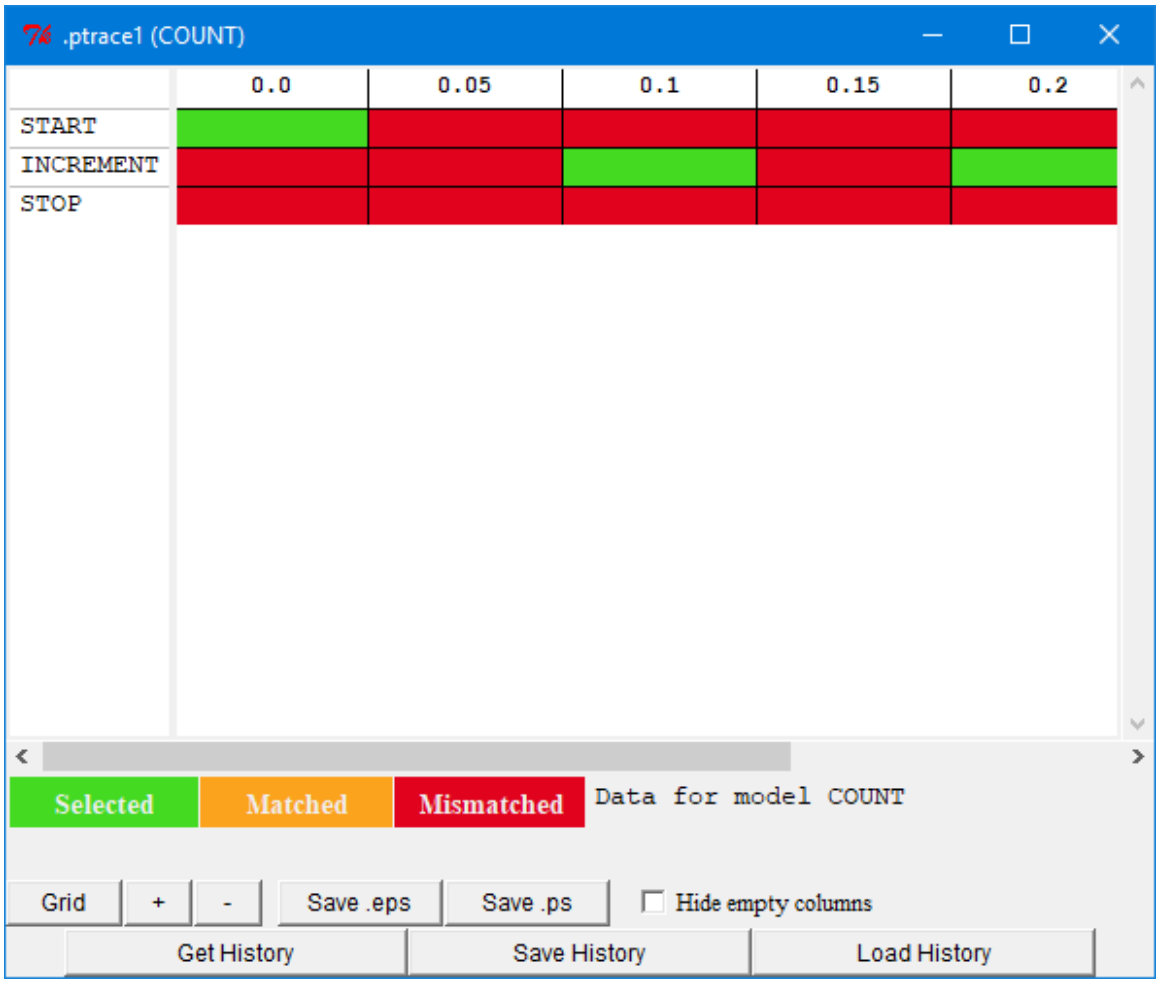
There are two views available for the Production history information: grid and graph. Which view to use is chosen using the menu button to the right of the “Production” button whichever value is shown on that menu button will be the one that is opened when the “Production” button is pressed. Both views use the same history information. Therefore if a window for either type is open the data will be recorded and the other type may be opened later to view that data.

Grid

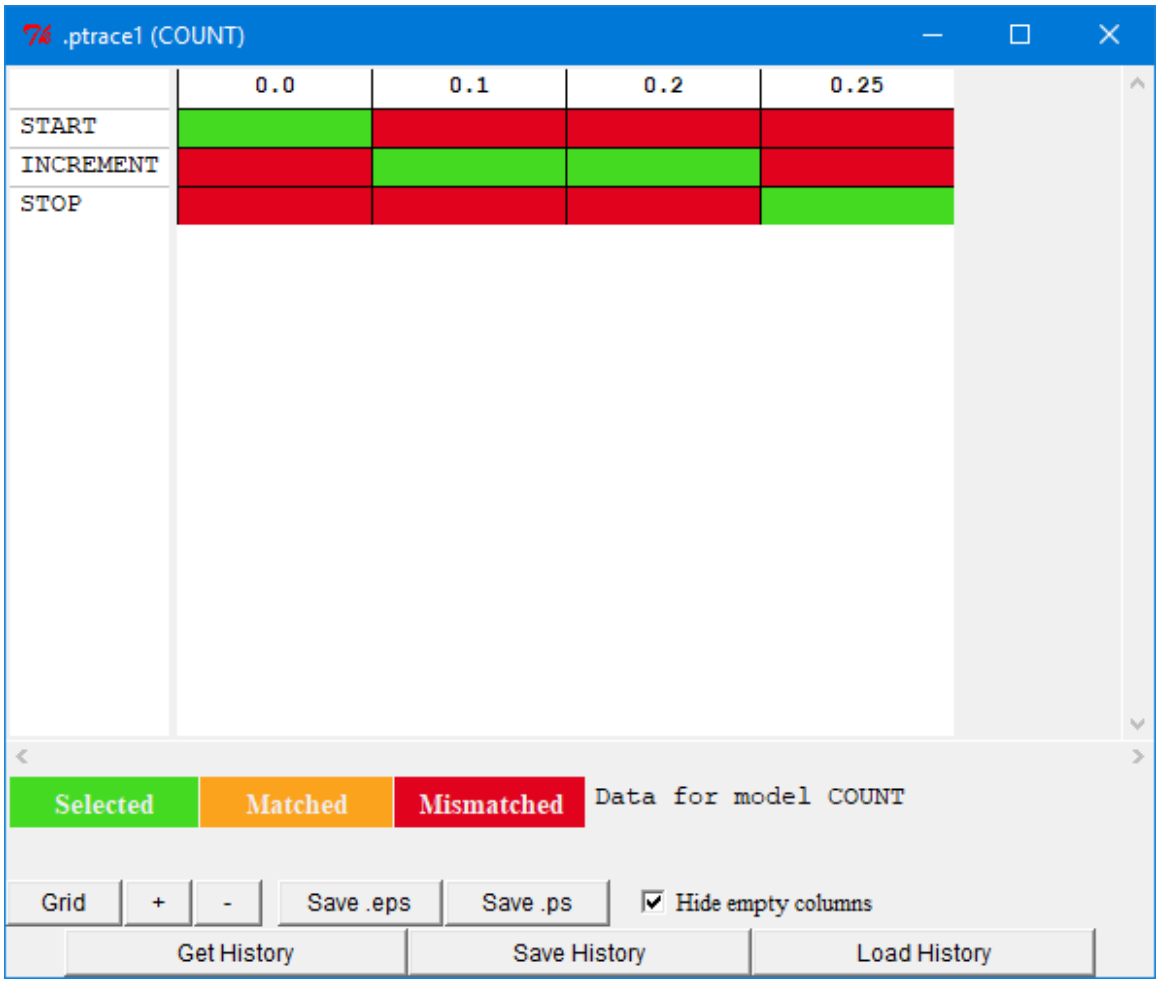
The Production grid tool can be used to display the information for each conflict-resolution action performed by the procedural module to select a production. Here is a display of the window without any data shown (how it will always appear upon opening):



Here is the initial data displayed after running the count model from unit 1 of the tutorial:



The left column displays all the names of the productions in the model, one per row, and if you click on the name of a production then it will open a new window showing the text of that production. To the right of the production names there is a column for each time that there was a conflict-resolution event in the model with the time of that event in seconds listed at the top (times increasing to the right). By default each conflict-resolution event will have a column. However, if you check the "Hide empty columns" box at the bottom of the window it will redraw the graph with the empty columns removed. Here is that same display after checking the "Hide empty columns" box:

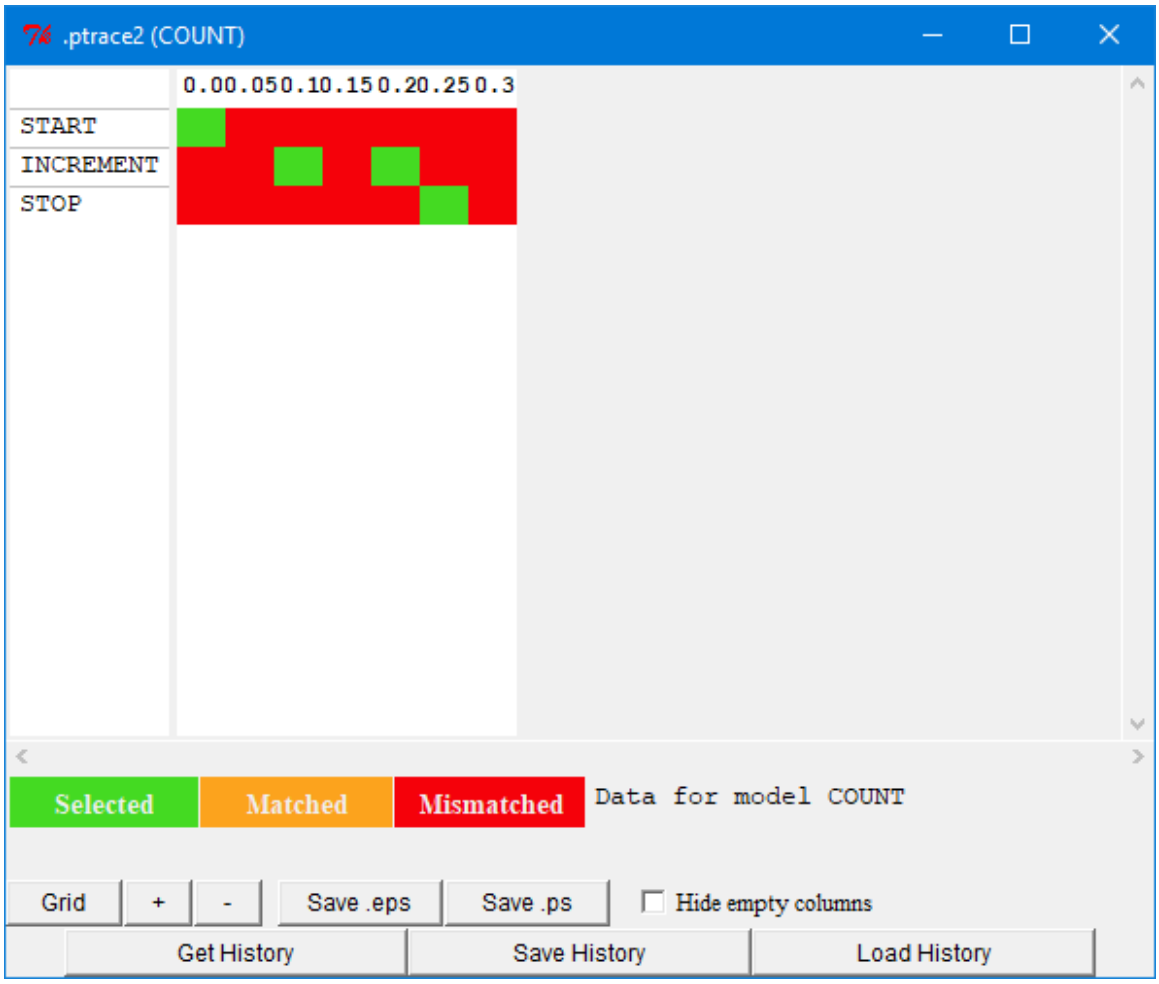


The color of the cell for a production's row in a column indicates whether or not that production matched during that conflict-resolution event and whether or not it was the production which was selected. The color coding of the cells is shown below the grid and by default if the cell is green, then the production matched and was selected. If the cell is orange then the production matched but it was not selected, and if the cell is red then the production did not match. The colors used can be changed by clicking on one of the colored boxes and then picking a new color in the color chooser that is opened. If a production was generated later in a run, typically through production compilation, then for the columns at the times before it existed it will not have any of those colors.

The scroll bar along the bottom of the display allows you to scroll through the history if it

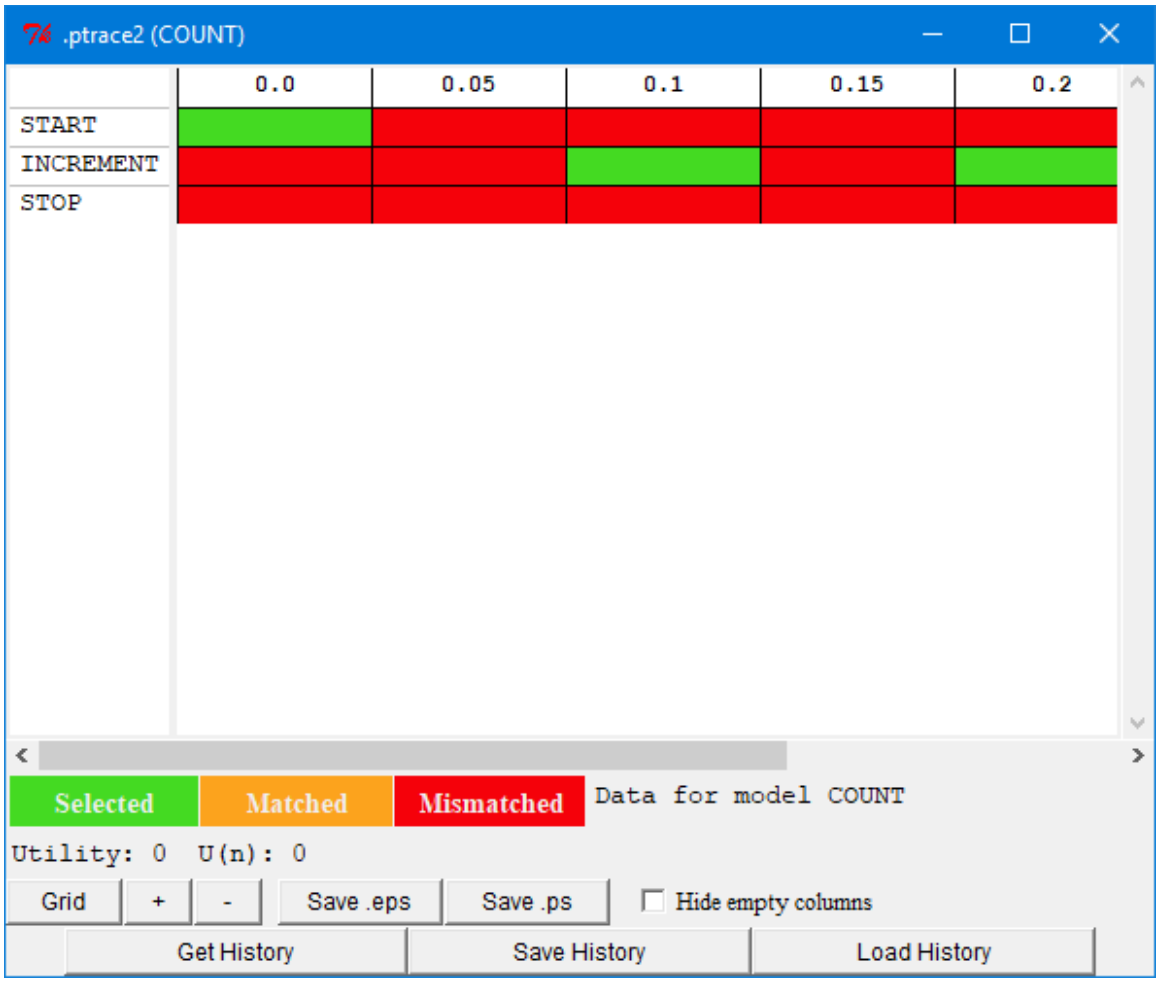
is wider than the display. The "+" and "-" buttons allow you to zoom in or out on the display, and the "Grid" button cycles through three options for whether or not the black grid lines are drawn for the columns and rows: both drawn, only the row lines, none of the lines.

Here is the same data with the grid lines removed and zoomed out multiple times:

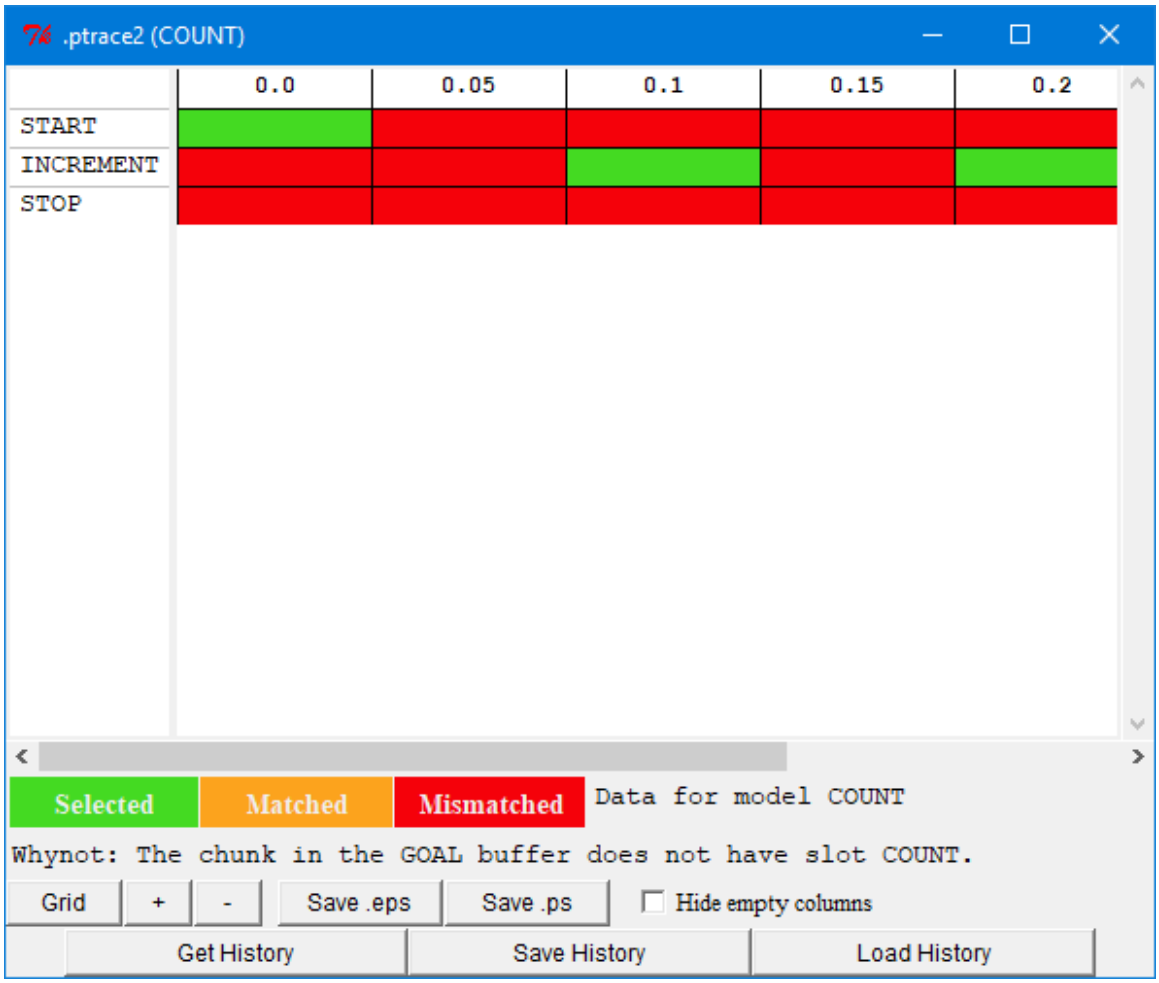


Placing the mouse cursor over the cells in the display will result in additional information being displayed above the buttons in the window. If the cell is selected or matched, then both the noisy utility value of that production at that time (which is what determined which one was chosen) and the true U(n) value of the production at that time will be displayed. Here is that display with the mouse over the start cell at time 0.0 (this model

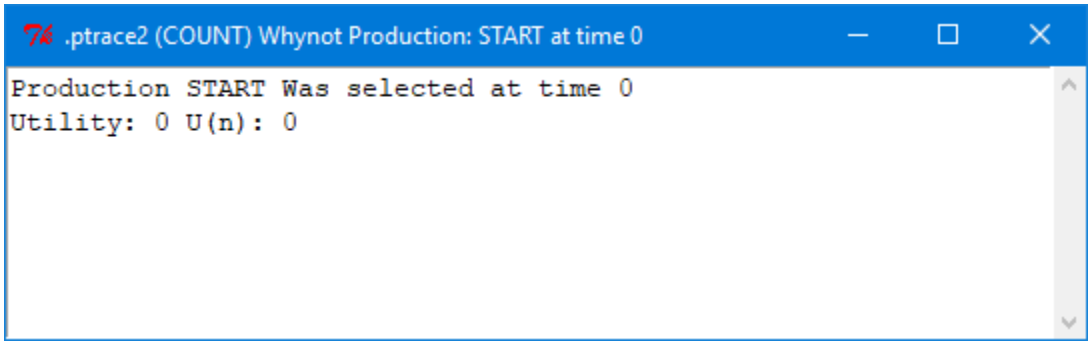
does not have utility noise enabled):

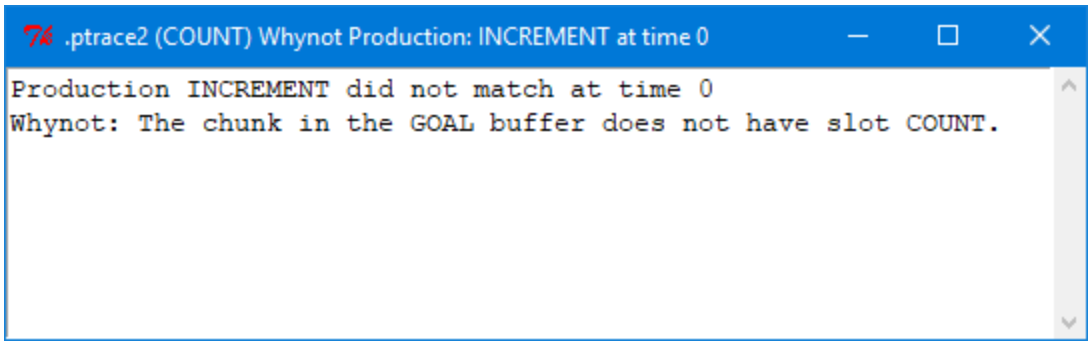


If the mouse is placed over one of the mismatched cells then the reason returned from the ACT-R **whynot** command at that time is displayed to indicate why that production did not match. Here is the display with the mouse over the increment cell at time 0.0:



If a cell in the grid is clicked then a separate window will open to display the Whynot information, which might be more useful if there is a lot of text. Here are examples from clicking those same two cells:





The “Save .eps” and “Save .ps” buttons at the bottom of the display can be used to save an image of the entire production matching grid. The “Save .eps” button saves an image of the graphic trace encoded as an Encapsulated PostScript file as a single page graphic. The “Save .ps” button saves an image of the graphic trace as multiple page (if needed) PostScript file in landscape mode.

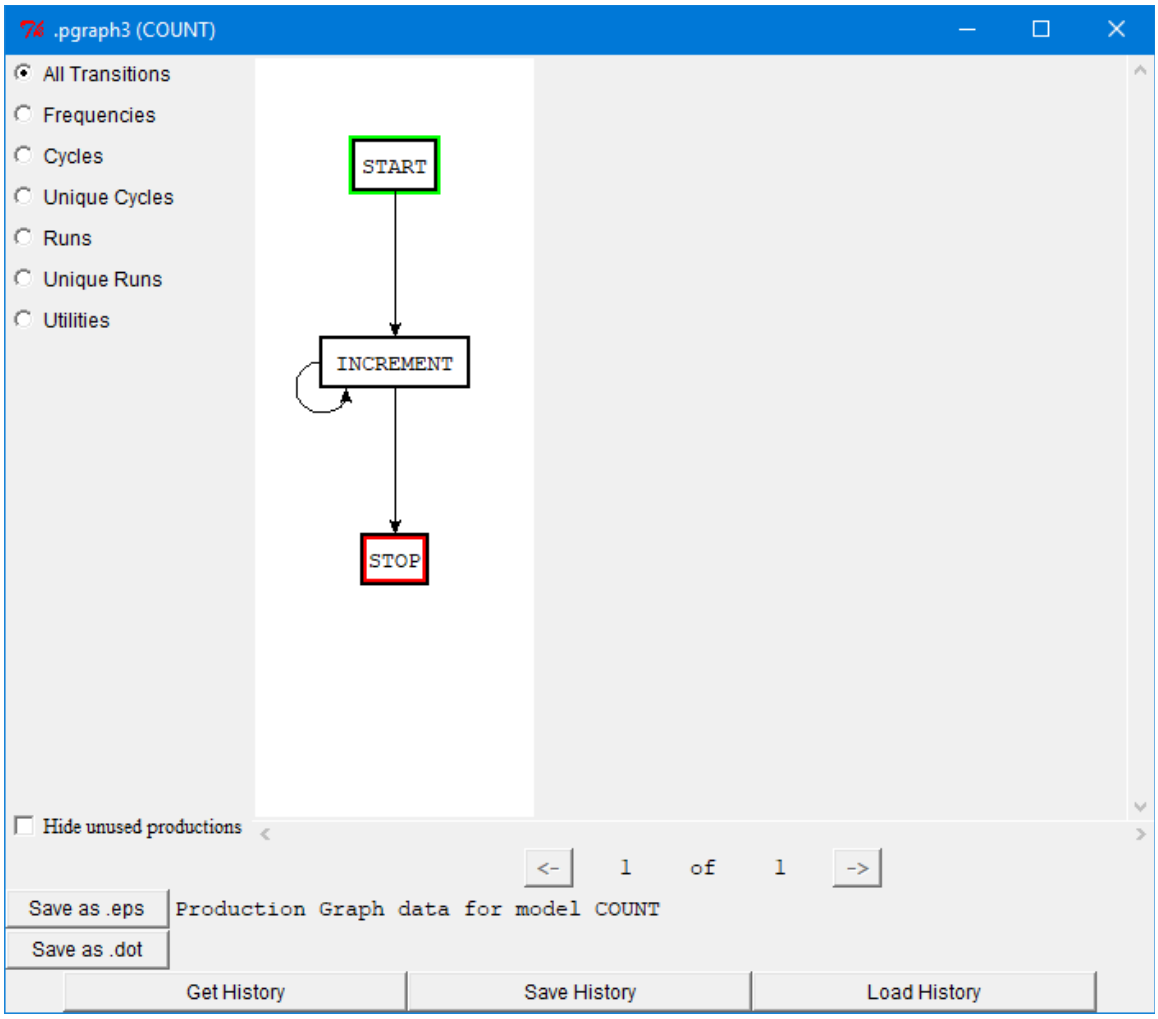
One helpful use for this tool is in investigating the productions which are generated through production compilation. Often many new productions are generated and it can be difficult to determine which ones are becoming generally useful or which ones are never being used. This tool would provide some graphic feedback to help locate the learned productions which are never matching and those which are matching and being selected often.

Graph

The Production graph tool provides a graph of the production transitions which occurred in the model run. Here is a display of the window without any data shown (how it will always appear upon opening):

The data can be displayed in seven different formats based on the selections on the left: “All Transitions”, “Frequencies”, “Cycles”, “Unique Cycles”, “Runs”, “Unique Runs”, or “Utilities”. Each provides a slightly different view of the data. The similar operation will be described first, and then the specific details of each will be discussed. Choosing a different format after getting the data will cause it to be updated appropriately.

Here is the display after running the count model from unit 1 of the tutorial with the “All Transitions” format:



The display will be a state chart diagram of the productions in the model indicating the order in which they were selected and fired. Each production in the model will be drawn in a box. If the border of the box is black then that production was selected and fired at some time during the run of the model for which the data was recorded. If the box border is gray then that production was not selected and fired. [If the "Hide unused productions" box is checked then the gray boxes will not be displayed.] The box with the green highlight indicates the first production which was selected during the data period being displayed and the box with the red highlight was the last one selected. The arrows indicate the sequencing of the productions. An arrow from a production A to a production B means that production B was in the conflict set after production A fired. If the arrow is a solid black line then production B was selected and fired after production A, but if the arrow is dashed and gray then production B was not selected and fired even

though it did match the current state.

Some of the display formats have more than one graph, and to indicate that below the display it indicates which graph among the available ones is currently being shown. There will also be buttons with arrows on them to switch between which of the graphs is shown if there is more than one.

The “Save as .eps” button will save an image of the currently displayed production graph as an Encapsulated PostScript file.

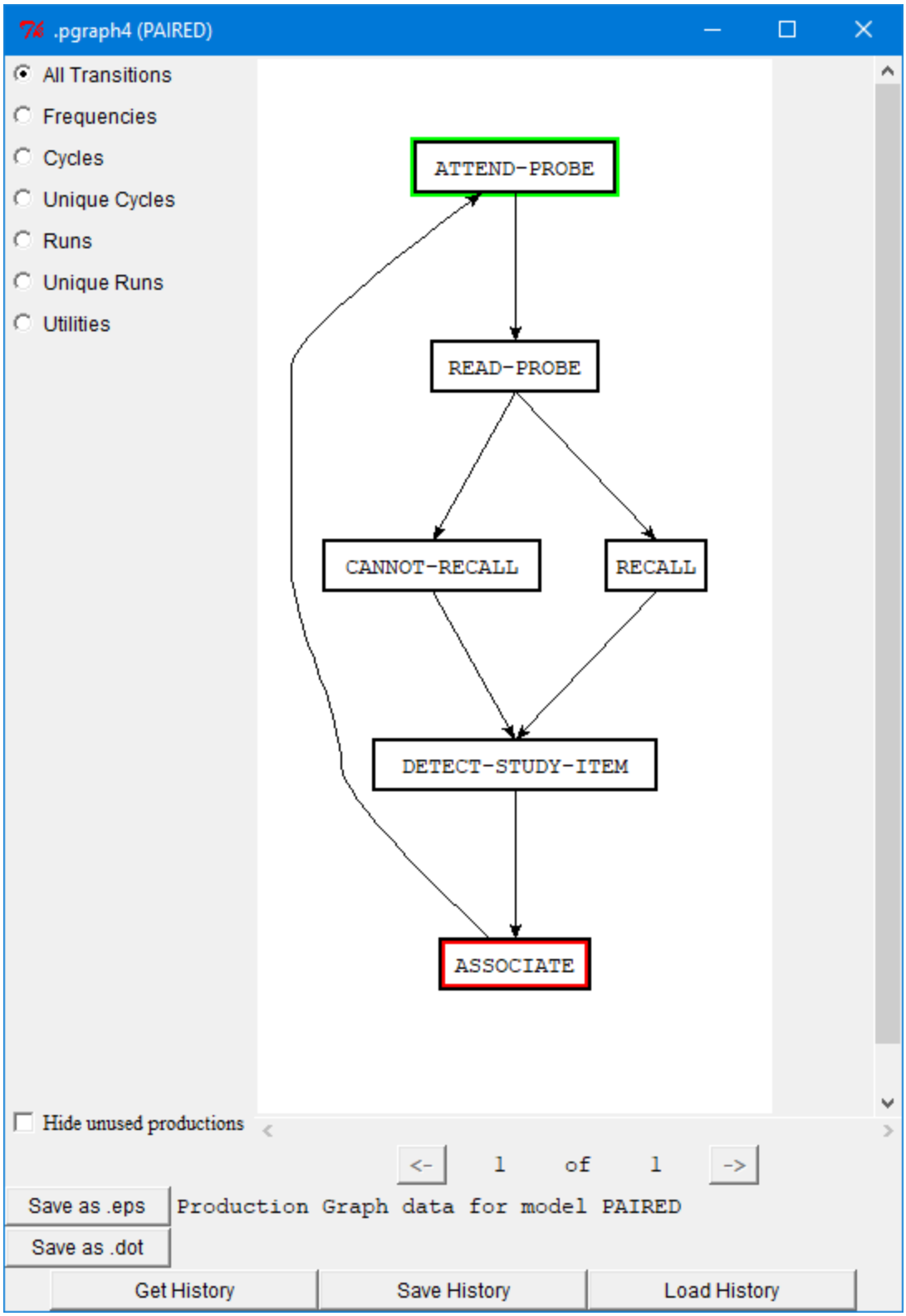
The “Save as .dot” button will save a text description of the currently displayed graph in DOT format for use with Graphviz or other purposes.

Clicking on the name of a production in the display will open a new window displaying the text of that production.

The differences between the displays for each format will be described next and examples for most will be shown from running the paired model from unit 4 of the tutorial using the command (paired-task 1 4).

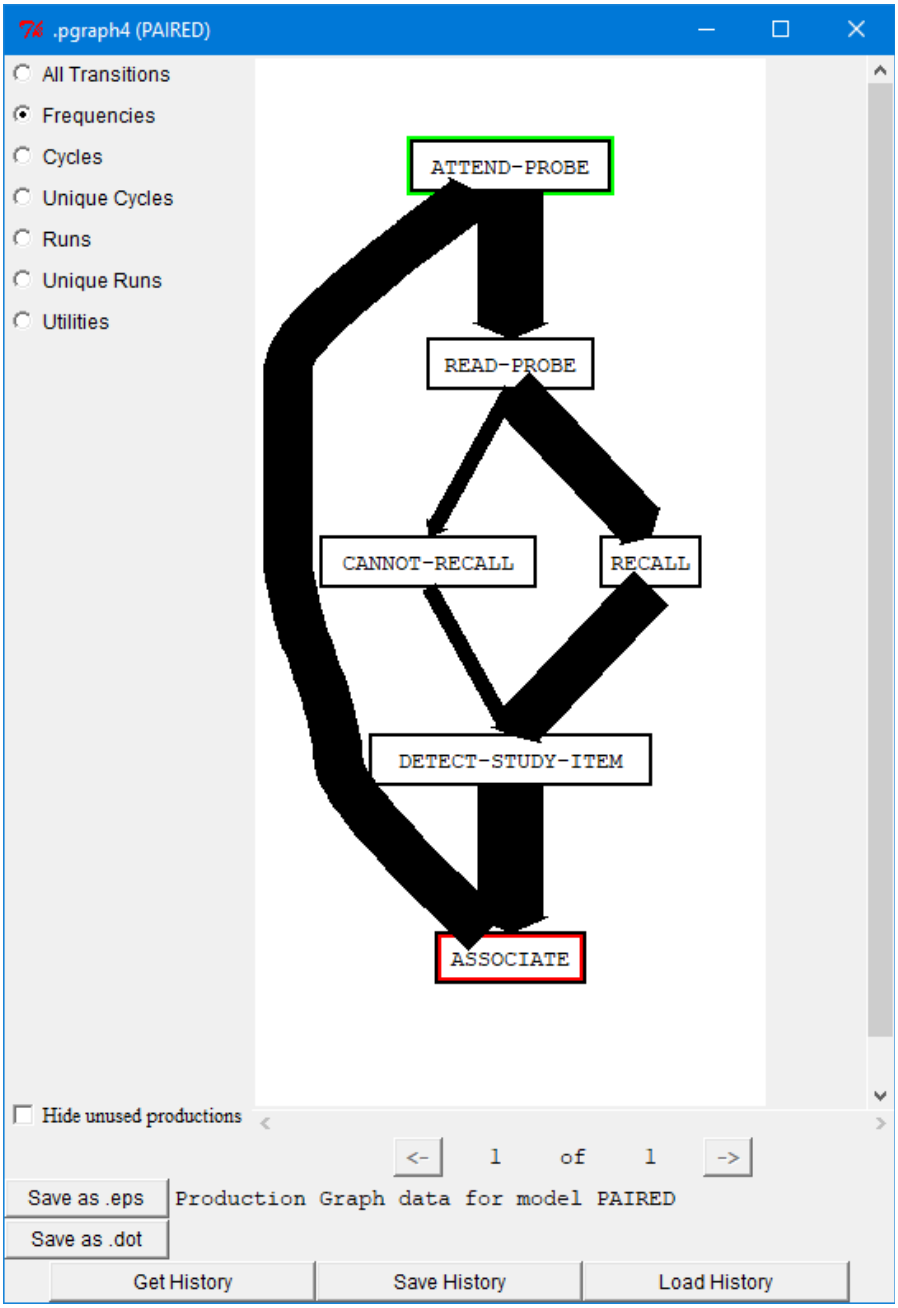
All Transitions

This format will always show only one graph. It will contain all of the transitions which occurred in the production data which was recorded. That data may involve multiple cycles in the graph (a loop which passes through a production multiple times) and there is nothing in the display to separate different cycles. Here is the display from the paired task:


.pgraph4 (PAIRED)
All Transitions
Frequencies
Cycles
Unique Cycles
Runs
Unique Runs
Utilities
ATTEND-PROBE
READ-PROBE
CANNOT-RECALL
RECALL
DETECT-STUDY-ITEM
ASSOCIATE
Hide unused productions
<-
1
of
1
->
Save as .eps
Production Graph data for model PAIRED
Save as .dot
Get History
Save History
Load History

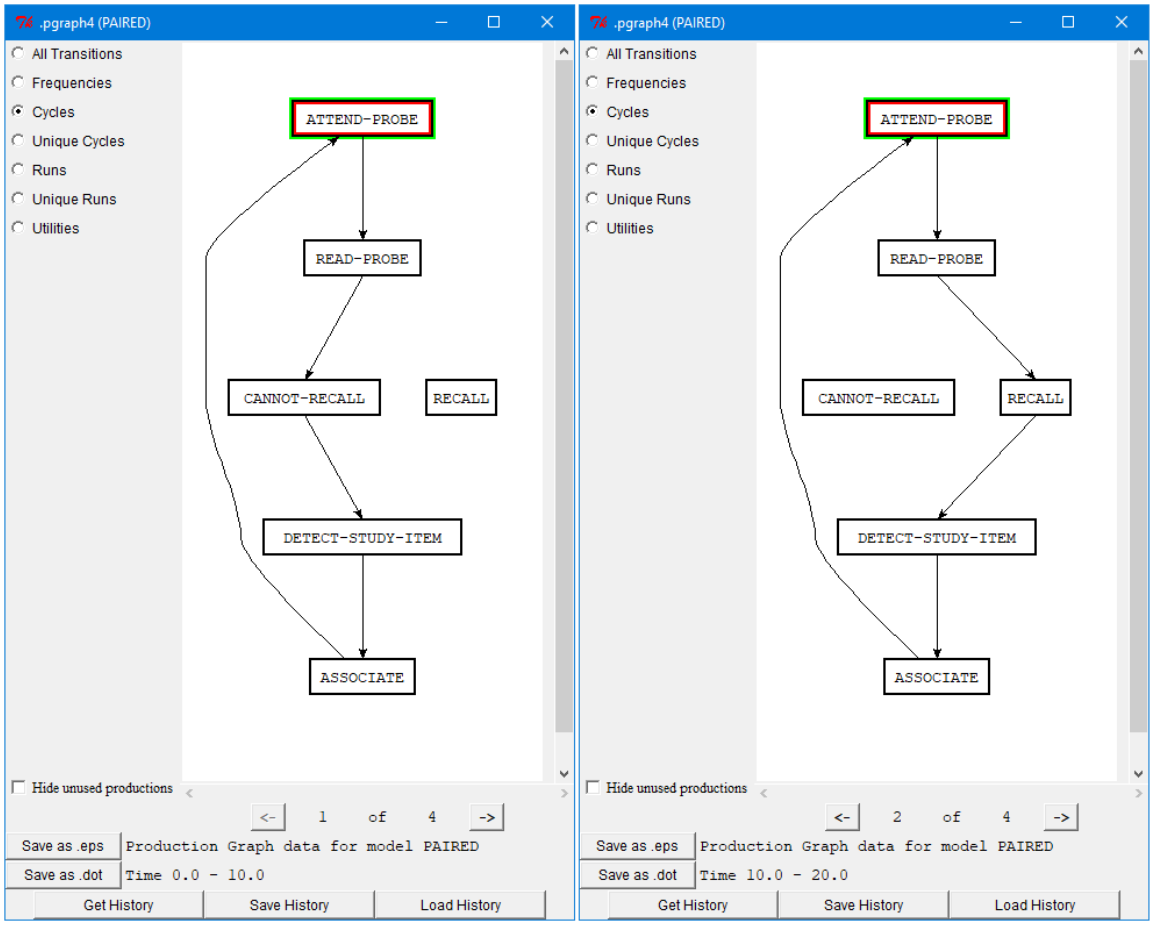
Frequencies

This format displays the same graph as the All Transitions format, except that the thickness of the links will indicate their relative frequencies with thicker lines being more frequent. Here is the result of that from the paired task run:



Cycles

This format breaks the data up into one display for each cycle which occurs (and possibly an incomplete cycle at the end if it does not form a loop). For each of the cycles displayed it will also show the model's time at the start and end of that cycle at the bottom of the window. Here are two of the Cycles from the example run:

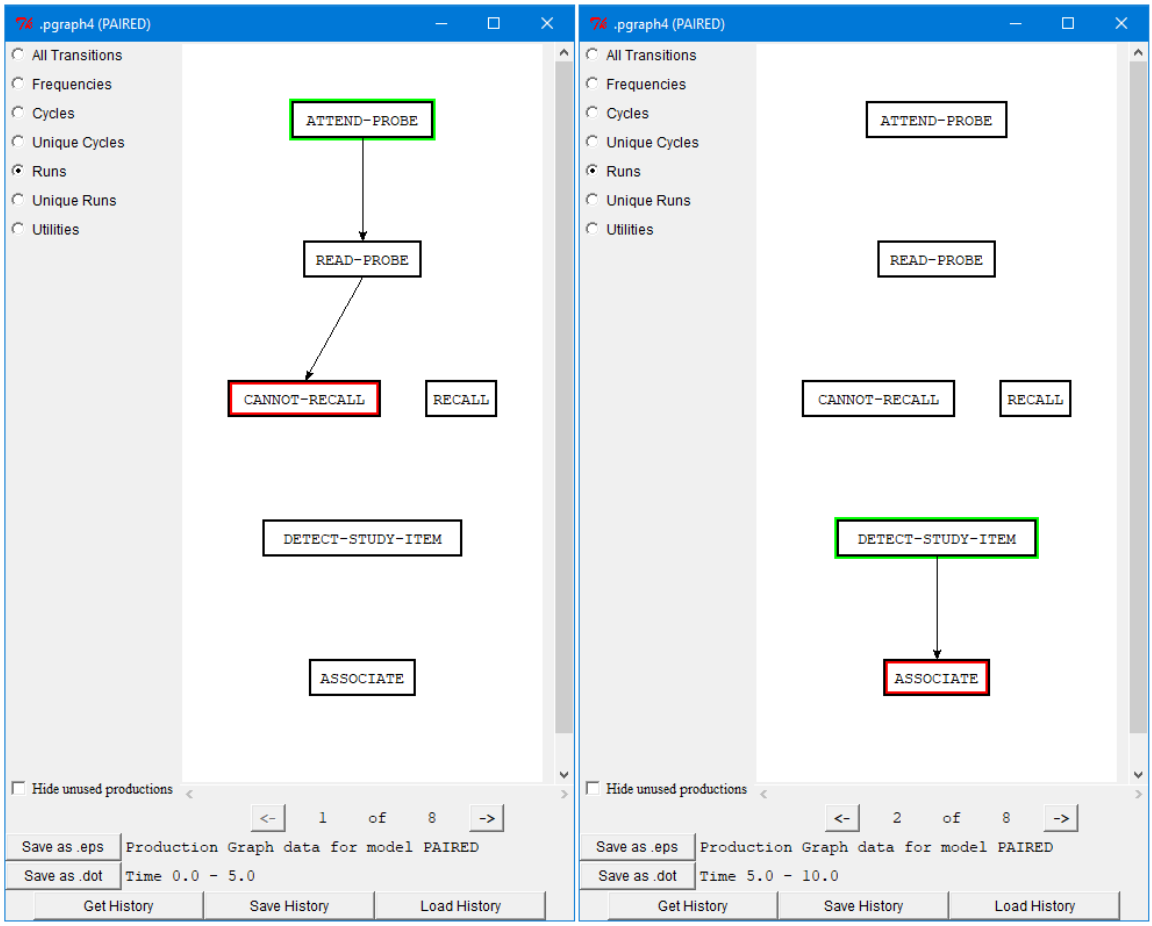


Unique Cycles

This format is similar to Cycles except that it only provides one copy of each different cycle which occurs in the data and does not provide the timing information. In the example run there are only 3 unique cycles among the 4 cycles of the data.

Runs

This format will break the data up into graphs based on when one of the ACT-R running commands is called, and will provide a separate graph for each run during which at least one production selection occurred. It will show the start and stop times for that run at the bottom of the display. Here are two of the Runs graphs from the example:

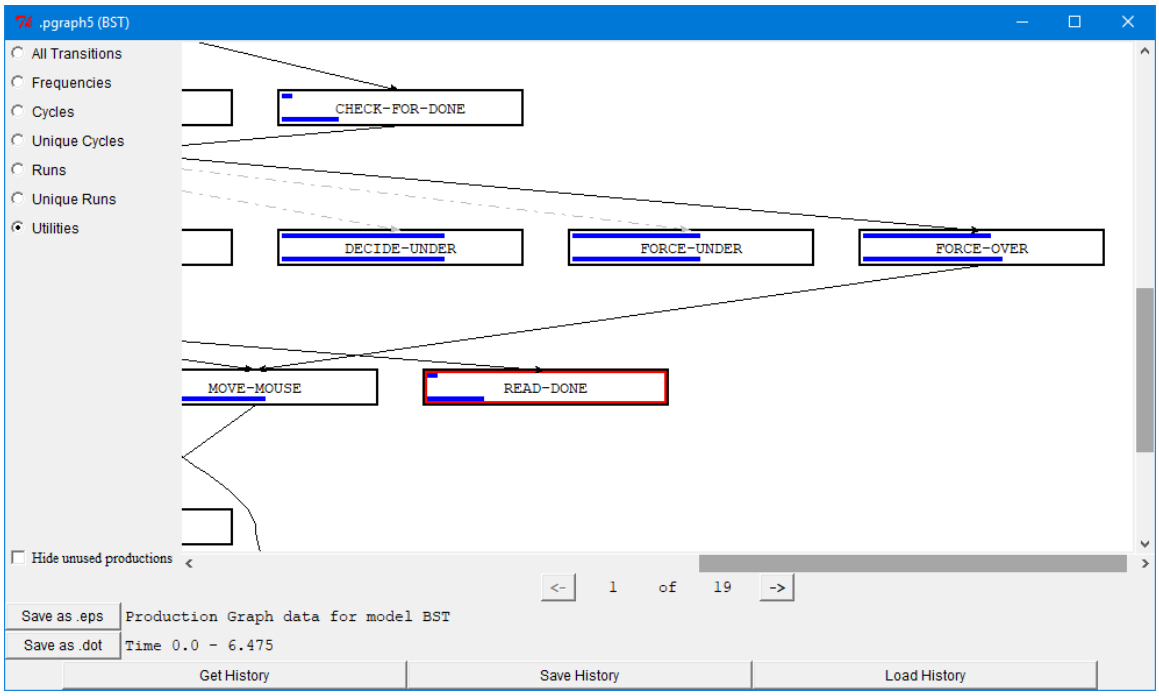


Unique Runs

This format is similar to Runs, except that it only provides one copy of each different run which occurs in the data and does not provide the timing information. In the example run there are only 3 unique runs among the 8 runs which occurred.

Utilities

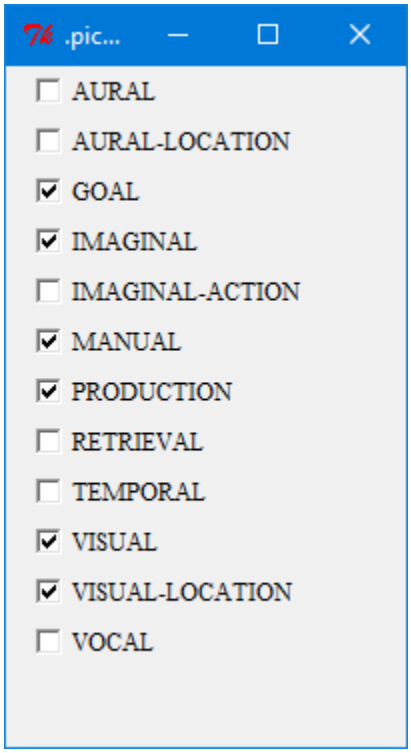
This format is similar to Runs except that the data is broken into graphs based on when the model receives rewards. Each reward marks the end of a graph, and there may be one additional graph at the end which does not represent a reward being presented if the model has fired additional productions after receiving the last reward. The display for the Utilities graph is slightly different than the others. First, all of the productions are represented in boxes of the same width instead of being sized based on the production names. In addition to that, each production box may have a blue bar displayed along both the top and bottom of the box. Those bars represent the relative utilities of the productions (the true utility not counting any noise which may have occurred during the run). The bar along the top represents the utility prior to the reward being provided and the bar along the bottom represents the utility after the reward has been propagated. The utilities are scaled across all productions and all graphs so that the maximum utility which any production has is represented by a bar which fills the box from left to right. Here is an example showing part of the graph after running the building sticks learning model from unit 6 of the tutorial. On this trial we can see that the force-over production was selected among the strategy selection productions and that its utility went up when read-done provided a reward.


.pgraph5 (BST)
All Transitions
Frequencies
Cycles
Unique Cycles
Runs
Unique Runs
Utilities
CHECK-FOR-DONE
DECIDE-UNDER
FORCE-UNDER
FORCE-OVER
MOVE-MOUSE
READ-DONE
Hide unused productions
<-
1
of
19
->
Save as .eps
Production Graph data for model BST
Save as .dot
Time 0.0 - 6.475
Get History
Save History
Load History

Buffer Based Recordable Data

The tools in this section are very similar to those from the General Recordable Data section and have the same common functionality. The difference between the tools in this section and those in the previous one is that for these tools one must also indicate specific buffers for which the data is to be recorded.

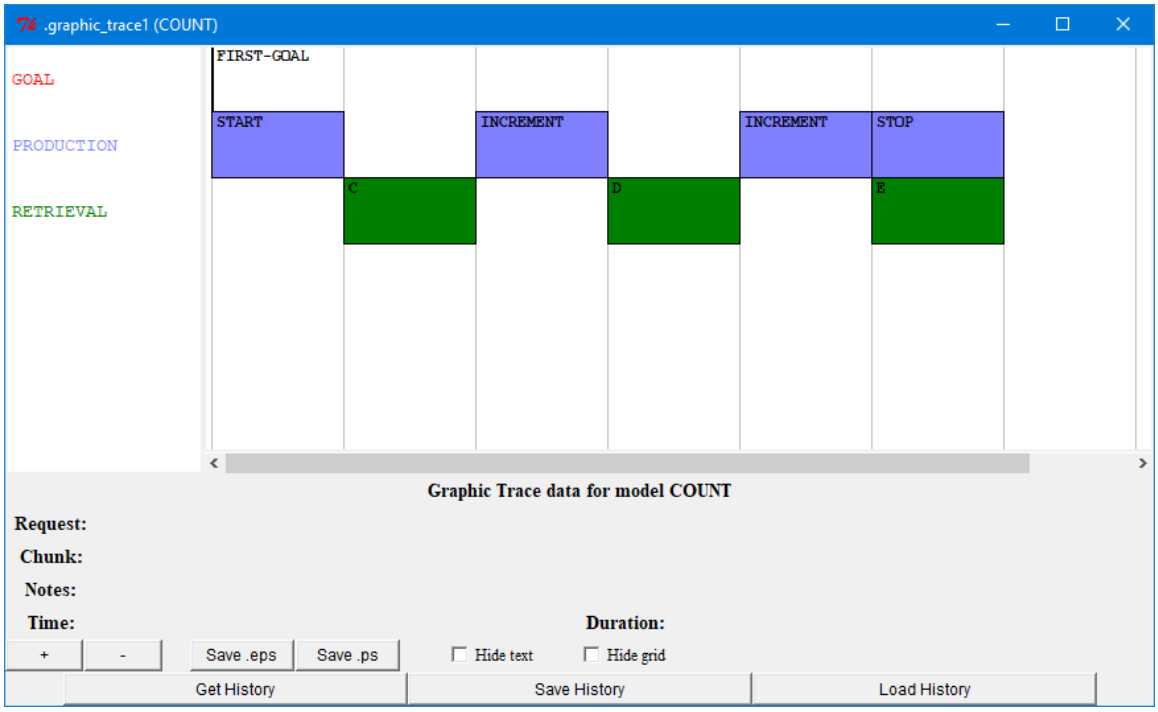
Opening any of the tools in this section will also open a window which shows all of the buffers in ACT-R along with a check box to select whether the data for that buffer should be recorded or not. That window will look like the one below and by default it will have all of the buffers which are used by the productions of the current model already selected:

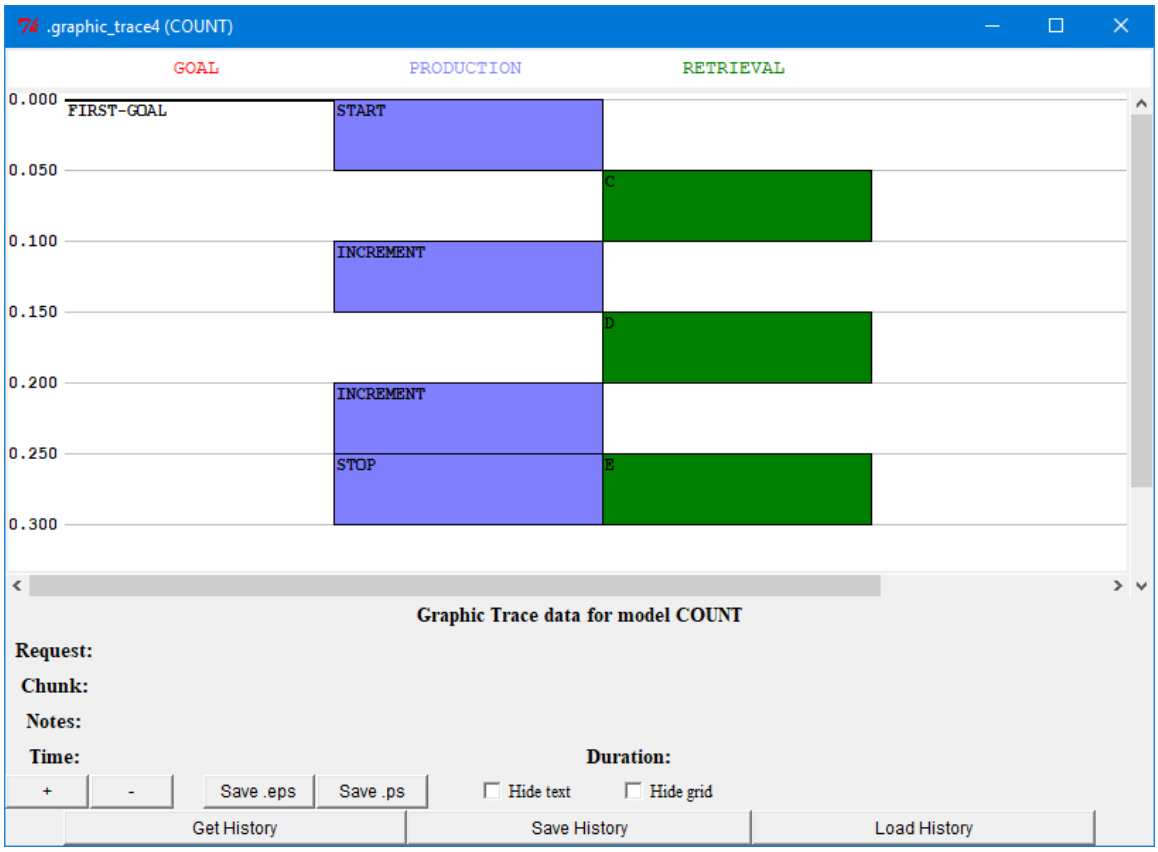


All of these tools rely upon the same history data stream (the buffer-trace) and thus changing the buffers which are recorded will affect the data for all of the tools. That buffer window can be closed after selecting the desired buffers.

Graphic Trace

The Graphic Trace tool provides a graphical representation of a model's operation showing activity for the buffers. It has two modes in which the data can be displayed, horizontal or vertical, and which to display is chosen using the menu button to the right of the "Graphic Trace" button. The two views work similarly and only differ in the orientation of the data. Any number of either type of window may be open at any time. Here are the horizontal and vertical traces from running the count model from unit 1 of the tutorial with the default set of buffers for that model being traced (production, goal, and retrieval):

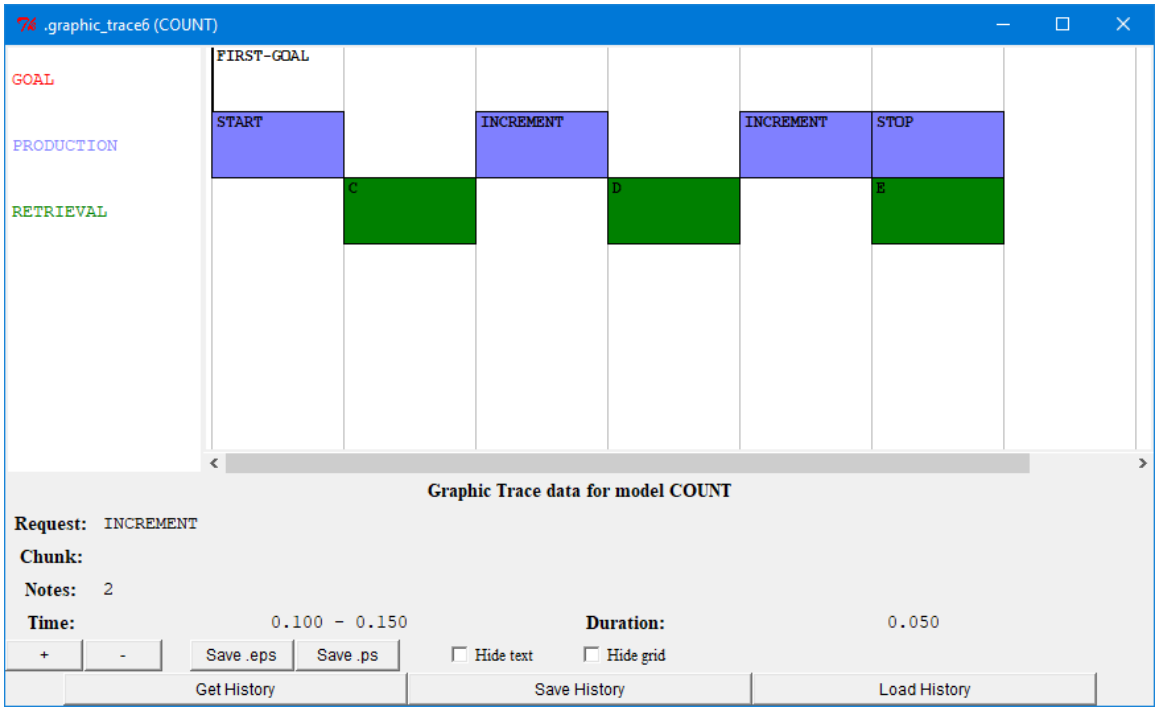


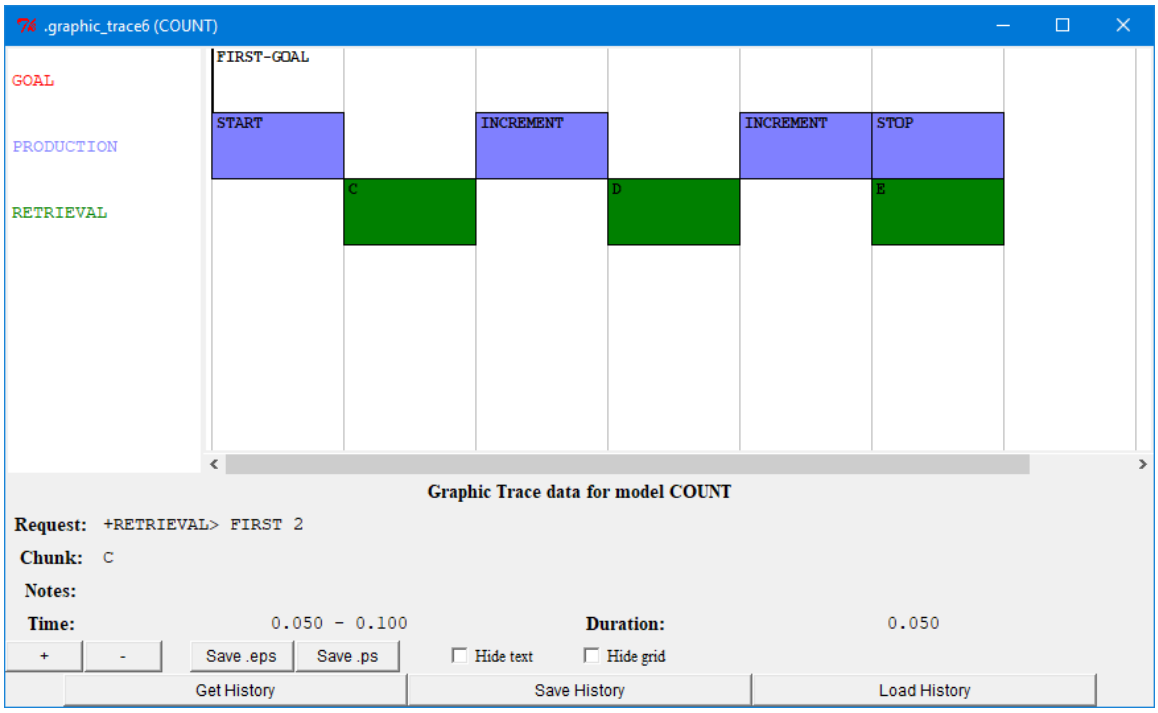


Each row of the horizontal trace or column of the vertical trace corresponds to one of the buffers in the model. The time runs along the bottom of the horizontal trace and along the left edge of the vertical trace. Boxes in a row or column indicate a time period during which that particular buffer reports that its module was busy, and typically represents the module's processing of a request. The text in the box shows the name of the chunk in the buffer (if there is one) for all of the buffers except the production buffer. The production buffer shows the name of the production which fired at that time. Note that some actions may complete instantaneously (like the goal buffer at time 0) in which case the chunk name will be printed to the right of (or below) the line indicating the activity.

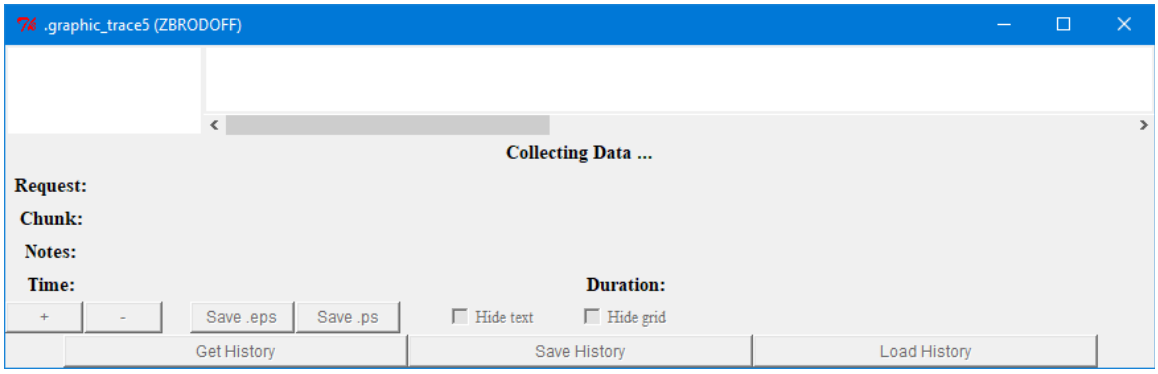
Placing the mouse cursor over a box in the trace will cause some additional details to be shown in the bottom of the display. The Time: and Duration: values will show for all activity. Time will display the start and stop time of that action in seconds, and Duration will provide its length in seconds. If the action was the result of a request then the Request: line will show the request which was made (or for the production buffer it just

shows the name of the production which fired). If the buffer holds a chunk at any point during that time (typically set at the end of a request) then the Chunk: line will show that. The Notes: line will show any information which has been recorded by the system or modeler using the add-buffer-trace-notes command. The only information which is currently added by the system itself is that the text produced by a production's !output! actions will show in the notes. Here are examples with the mouse placed over the second box in the production row and the first box in the retrieval row respectively:



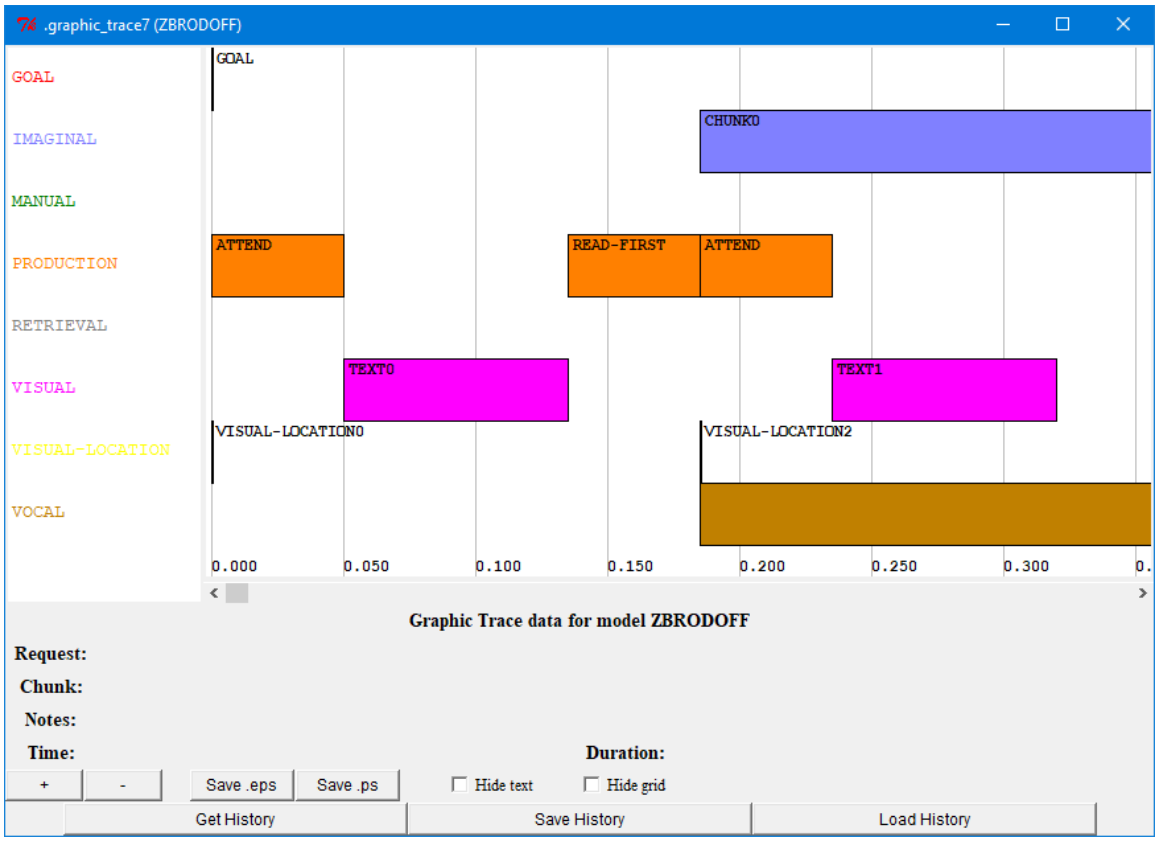


For a long trace it may take some time for the display information to be processed and drawn. While that is happening the status line will say "Collecting Data ...". None of the controls on the window should be used until that finishes.

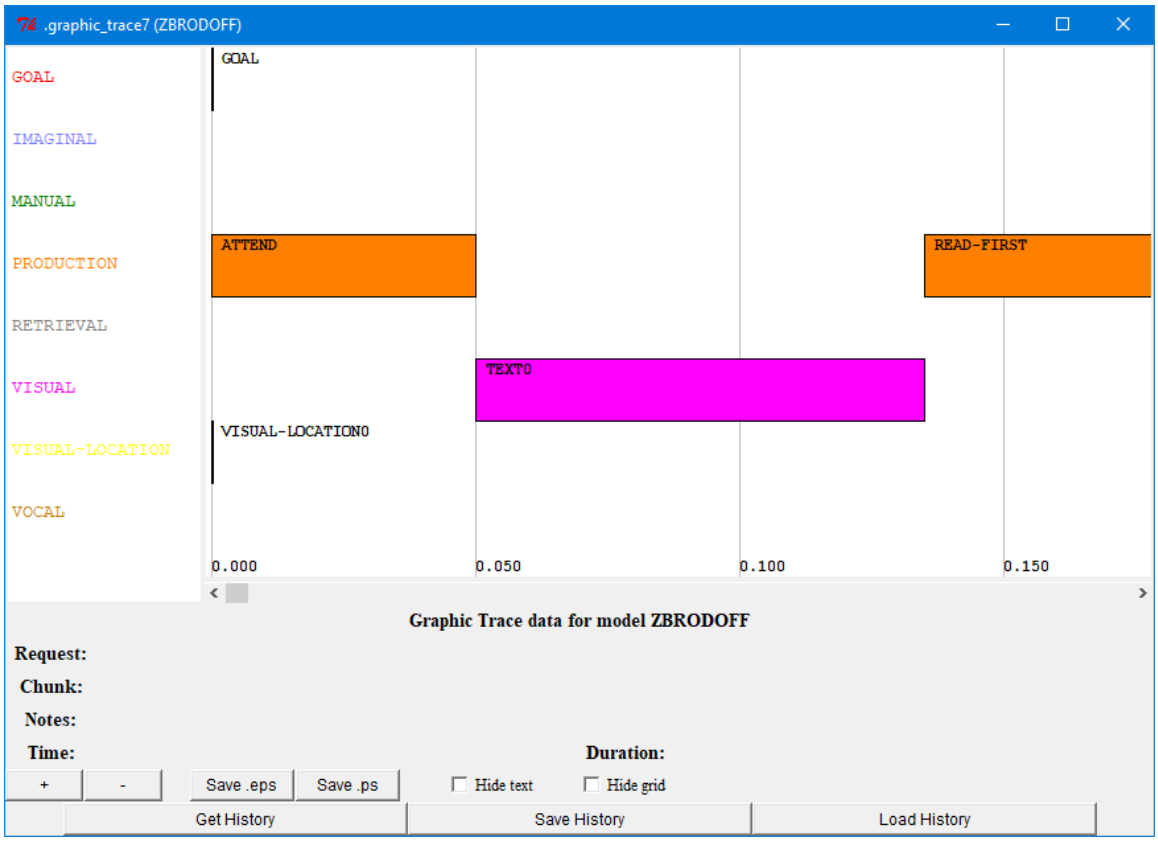


+ and – Buttons

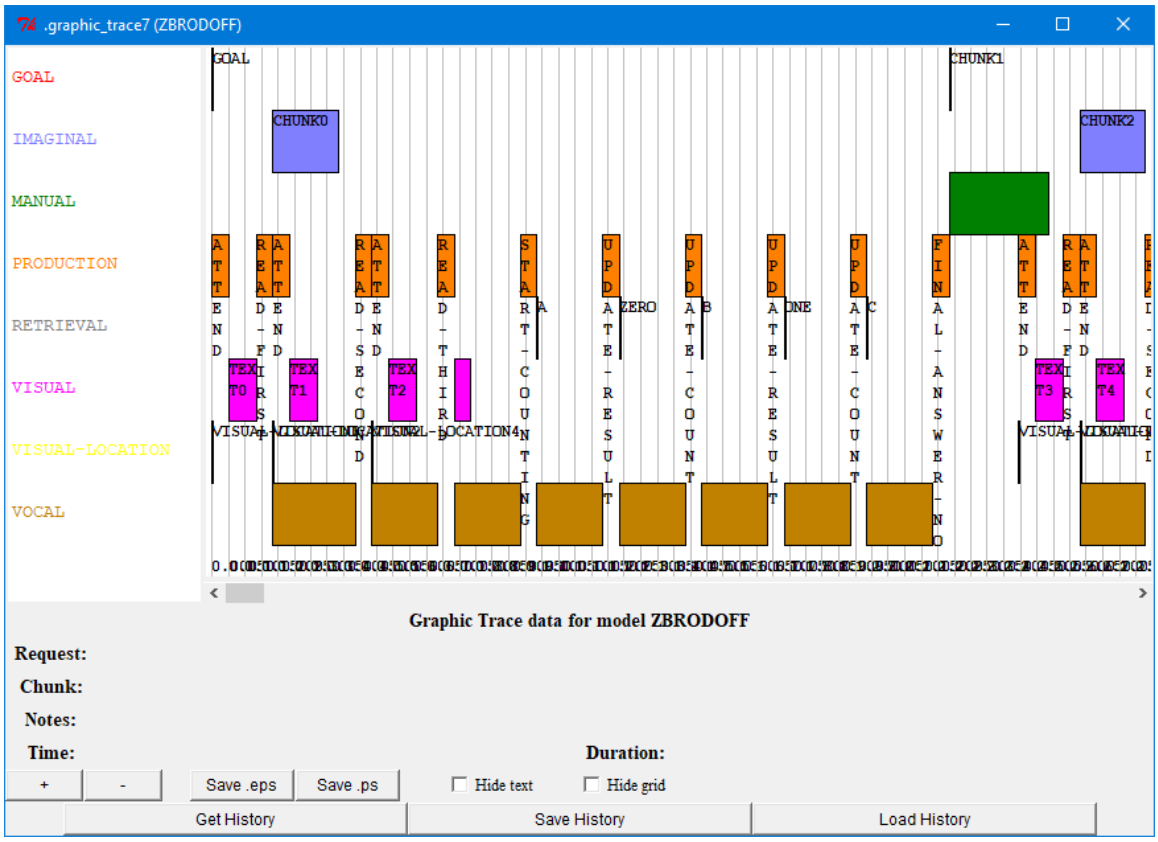
If the trace is larger than the window the scroll bars along the edges of the trace can be used to scroll the display, but the + and – buttons can also be used to zoom in and out on the display. Here is the default view of a run of one set of the zbrodoff task from unit 4 of the tutorial:



Pressing the + Button will zoom in on that display:

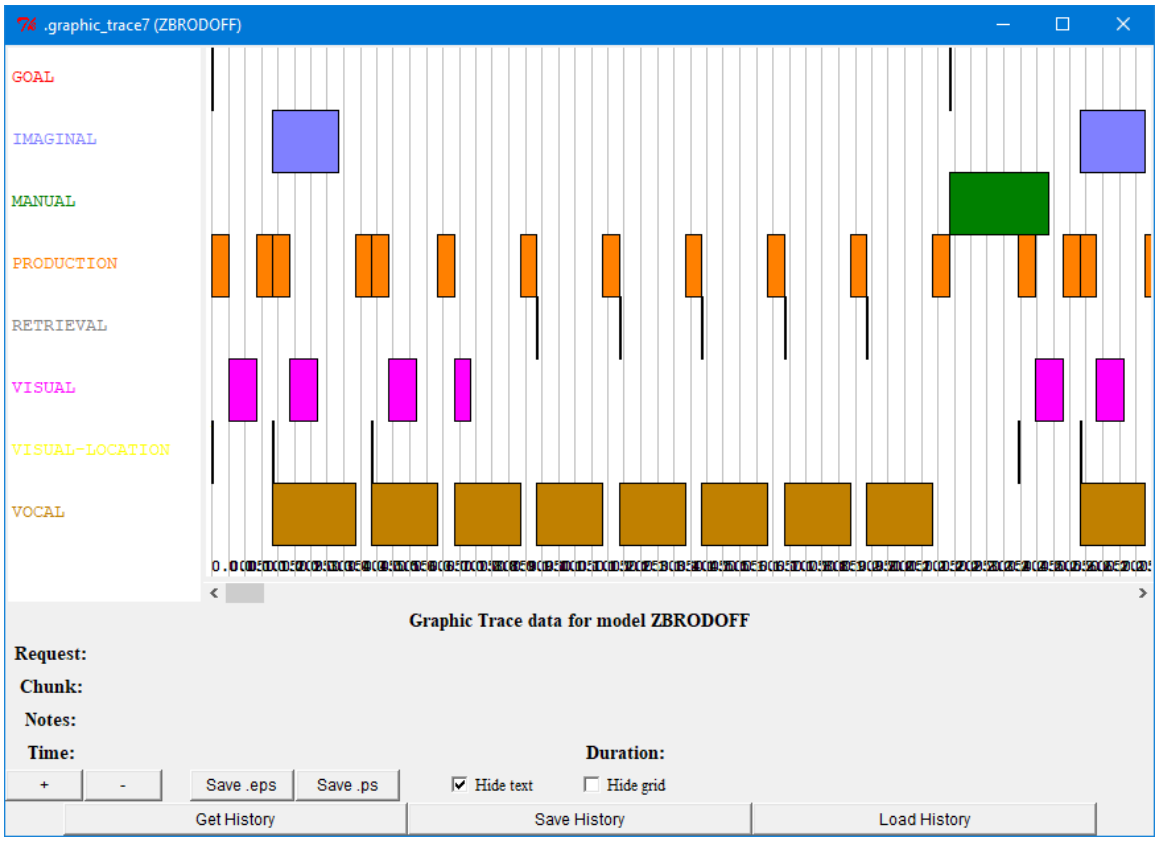


and pressing – multiple times will zoom out to something like this:

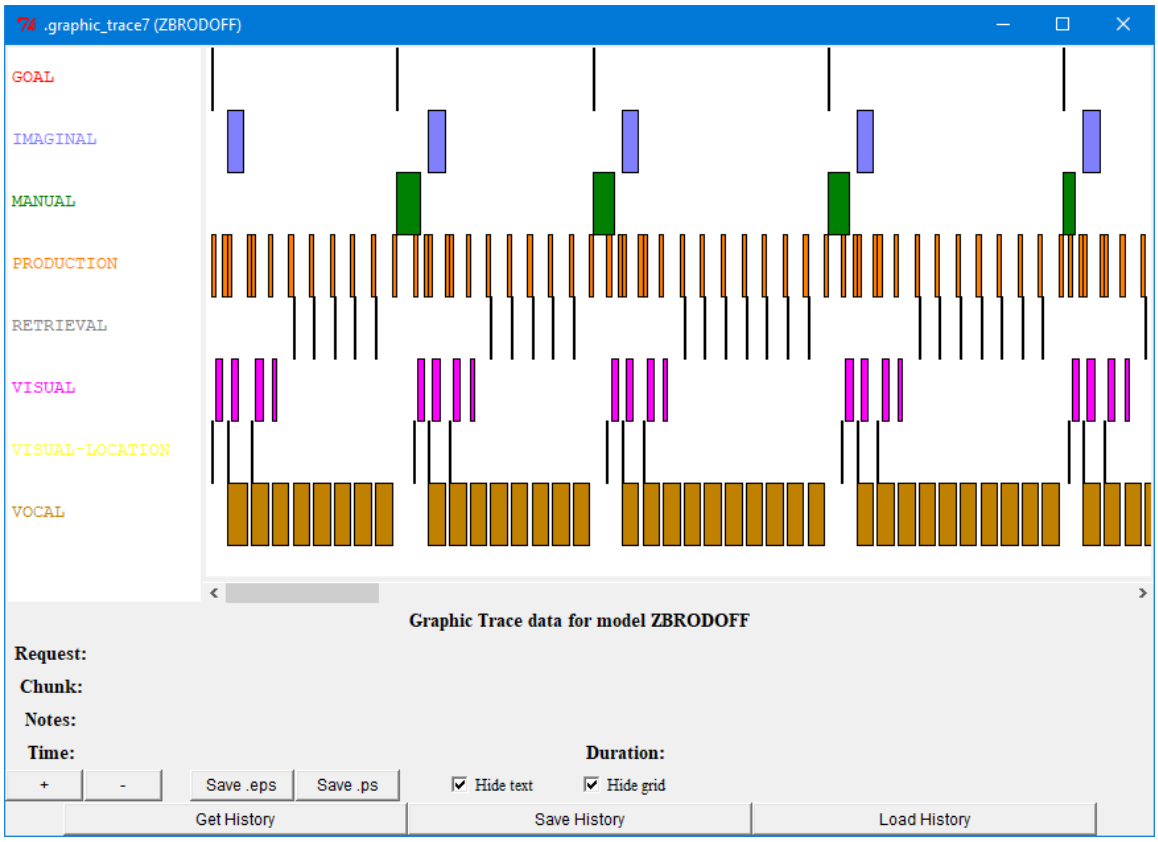


Hide text and Hide grid

When zooming out on a large trace to get a more general view of what is happening in the model, as shown in the zoomed out image above, the text output does not display well and isn't really helpful. If you check the "Hide text" box it will remove the text from the display and here's that same image with the text removed:



Similarly, the “Remove Grid” button will eliminate the grid lines with the time information from the display. Here is the same trace zoomed out even further without the text and grid:



Save .eps

The “Save .eps” button can be used to save an image of the graphic trace as an Encapsulated PostScript file. Pressing that button will bring up a file creation dialog in which you must provide the name for the file in which to save the data. The image will be saved as a single page graphic which contains the whole trace.

Save.ps

The “Save .ps” button can be used to save an image of the graphic trace as a PostScript file. Pressing that button will bring up a file creation dialog in which you must provide the name for the file in which to save the data. The image will be saved as a multiple page document. For the horizontal trace the page is generated in landscape mode, and for the vertical trace it is in portrait mode.

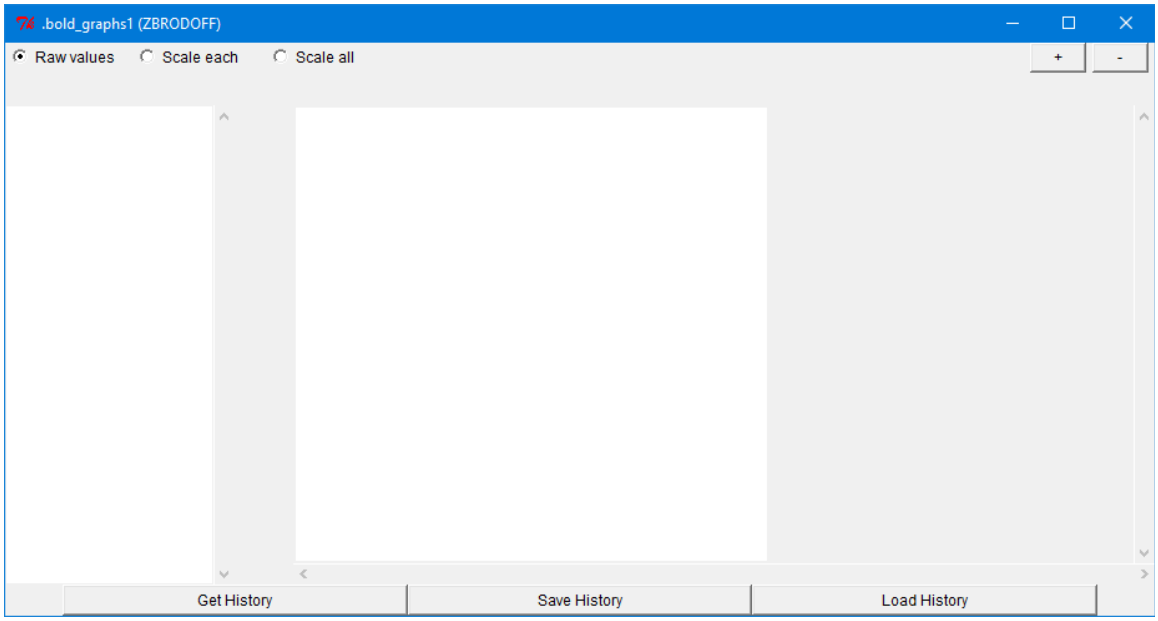
BOLD

There are three tools which can provide graphic representations of the BOLD (Blood Oxygen Level Dependent) response prediction data which a model generates. Which tool to open is chosen using the menu button to the right of the "BOLD" button. A full description of how that is computed is beyond the scope of this document, but a very brief description will be given here before describing the tools (more details can be found in the ACT-R Reference Manual).

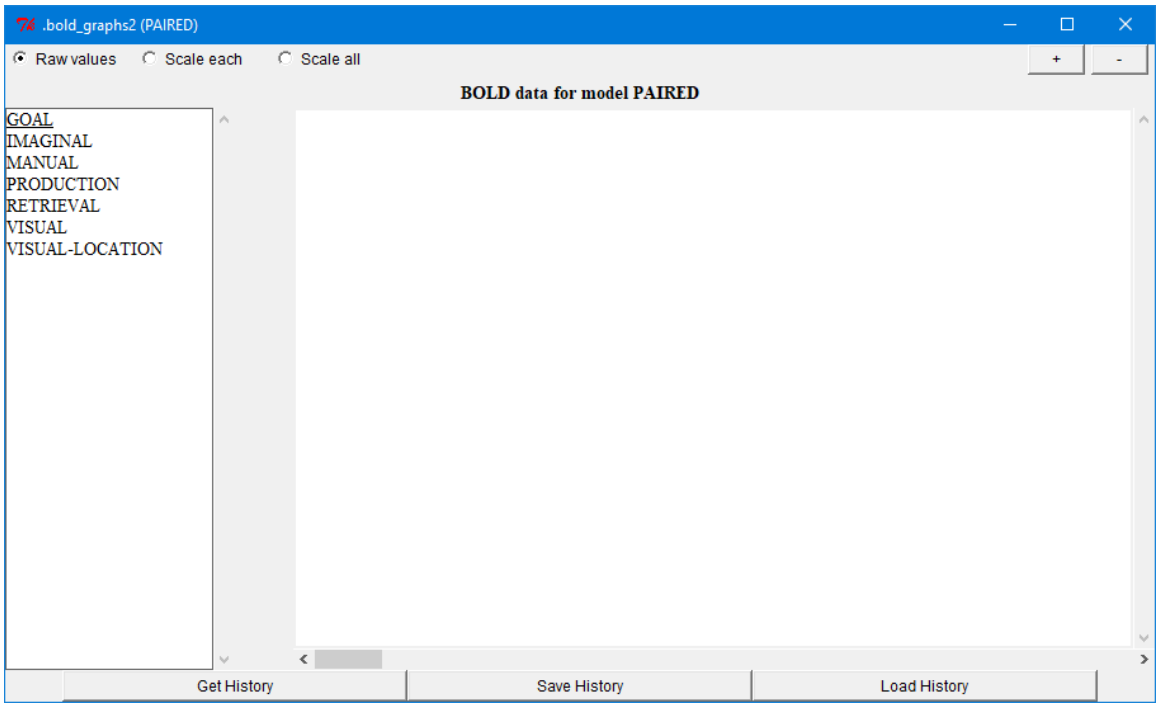
For each buffer in ACT-R the pattern of use, as shown in the graphic traces, can be recorded. That recorded pattern of use over a run can then be considered as a metabolic demand on the brain which can be combined with a hemodynamic response function to create a prediction of a BOLD response. Past research has lead to associating each of the buffers in ACT-R with a particular region of the brain. Thus, the patterns of use of the buffers lead to predictions for a BOLD response seen across various areas of the brain.

Graph

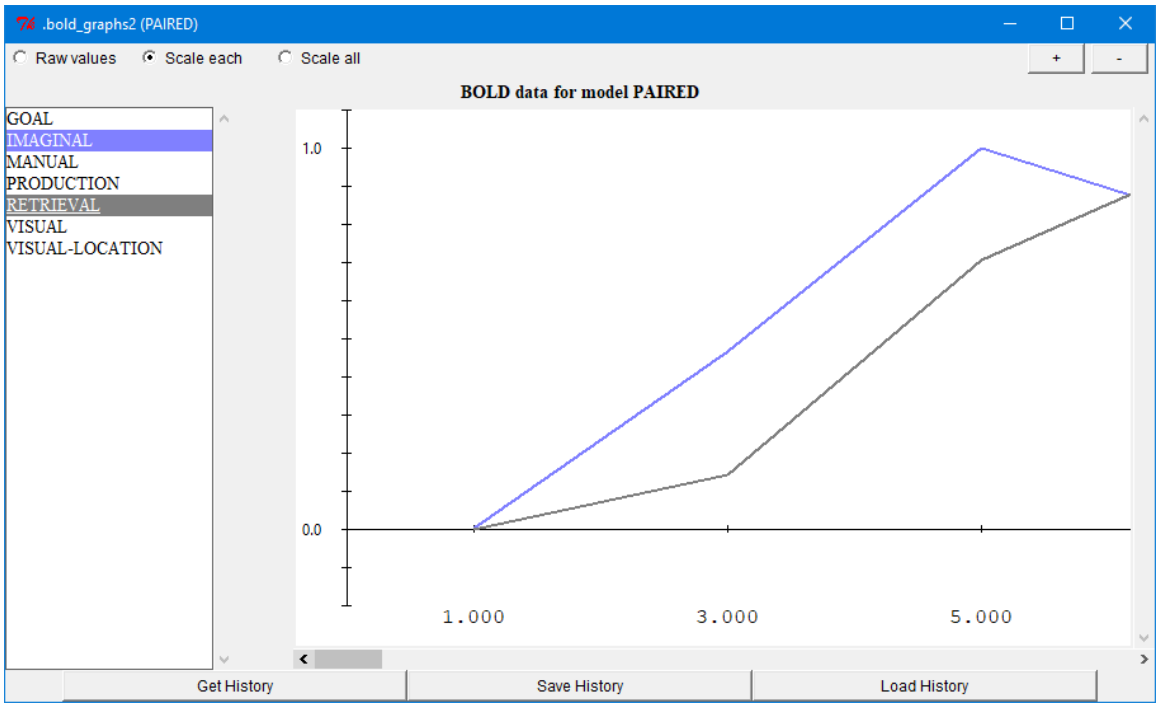
The graph display will draw a graph of the BOLD data for the buffers. Here is a display of the window without any data shown (how it will always appear upon opening):



When you get or load the data it will show the list of buffers which have data available on the left:

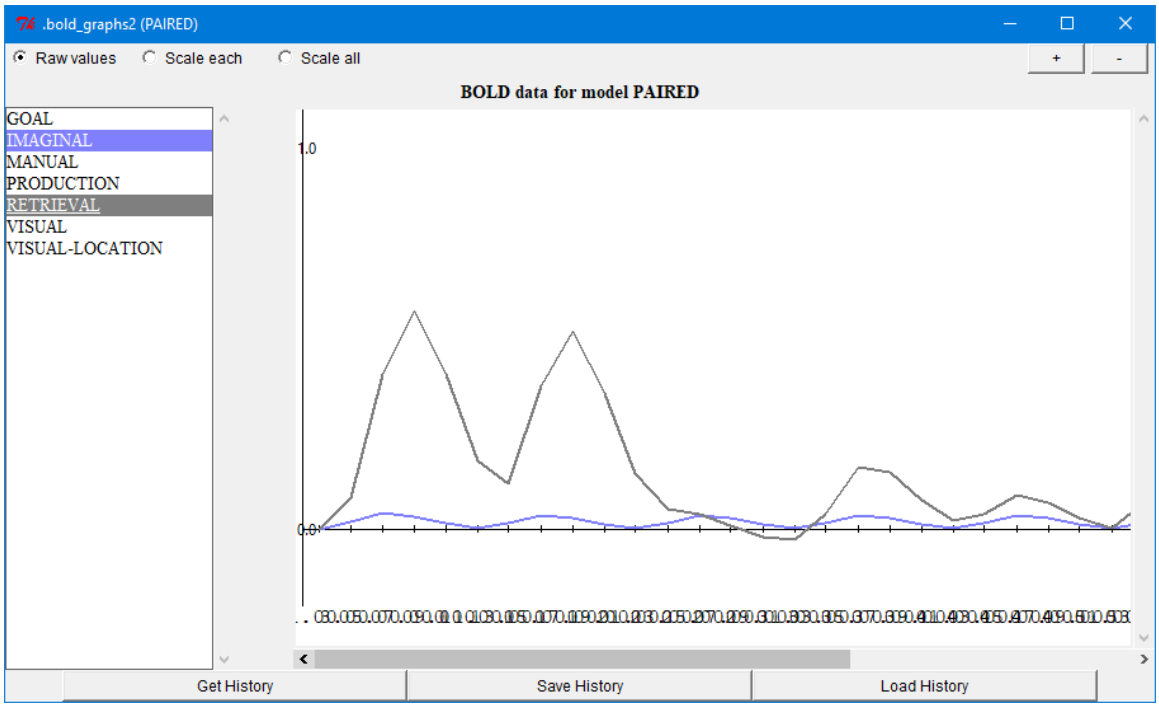


Selecting a buffer from the list will result in a graph being drawn in the pane on the right showing the data returned by the ACT-R **predict-bold-response** command for that buffer, and multiple buffers can be graphed at the same time. Here is the initial graph for the retrieval and imaginal buffers after running the paired model from unit 4 of the tutorial for trials with two items:



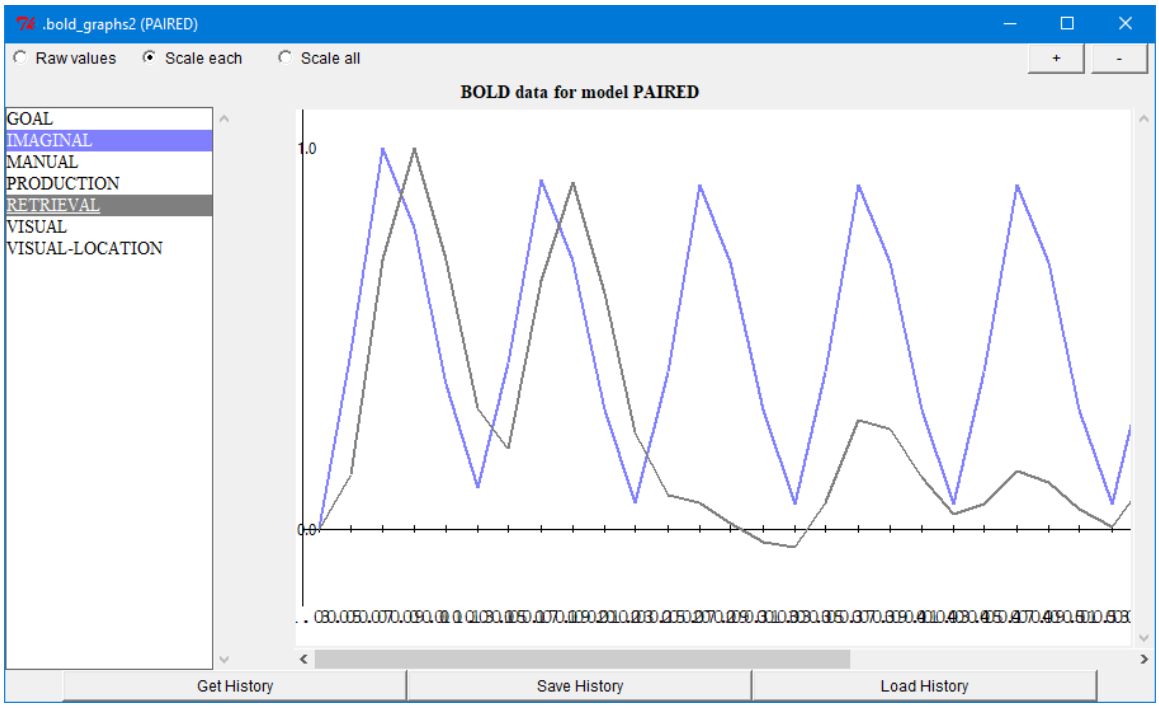
+ and – Buttons

The + and – buttons located in the upper right corner of the window can be used to zoom in and out on the display. Here is the same graph from above zoomed out a few times by pressing the - button:

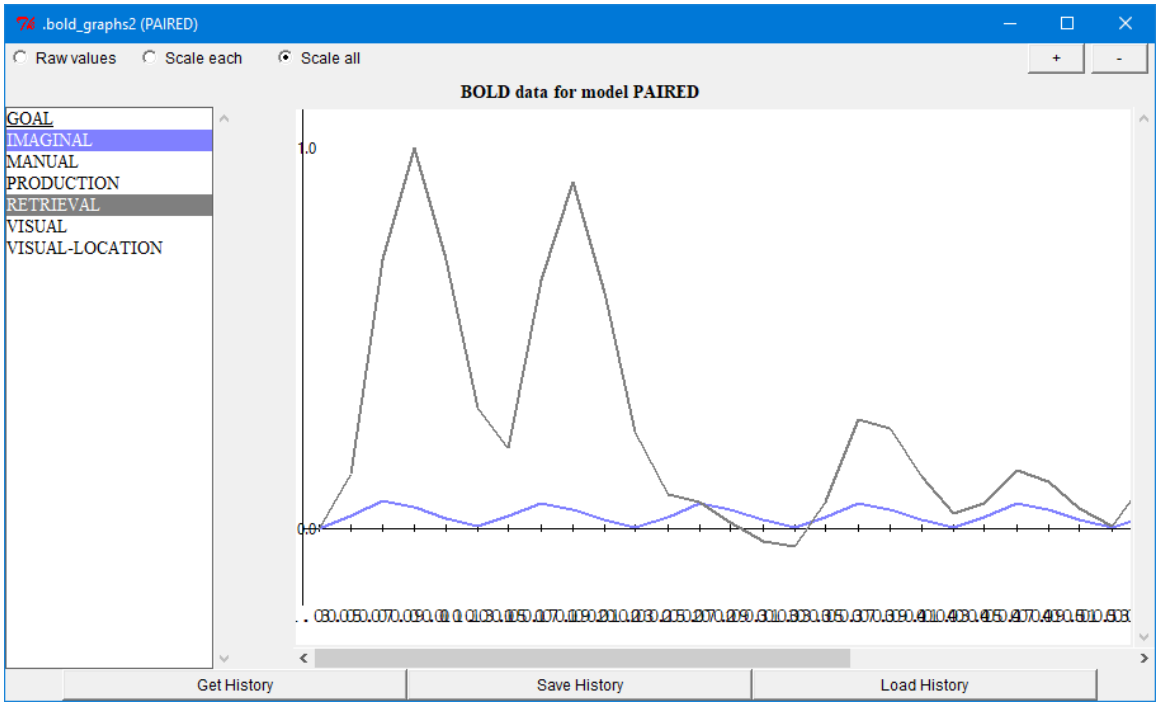


scaling the data

There are three options for how to scale the BOLD data which are selectable in the top left of the window: “Raw values”, “Scale each”, and “Scale all”. The default setting is “Raw values” which graphs the data as returned from **predict-bold-response**. Selecting “Scale each” will remap each buffer’s data so that it’s maximum value is mapped to 1.0. Here is that same data shown with “Scale each” selected:

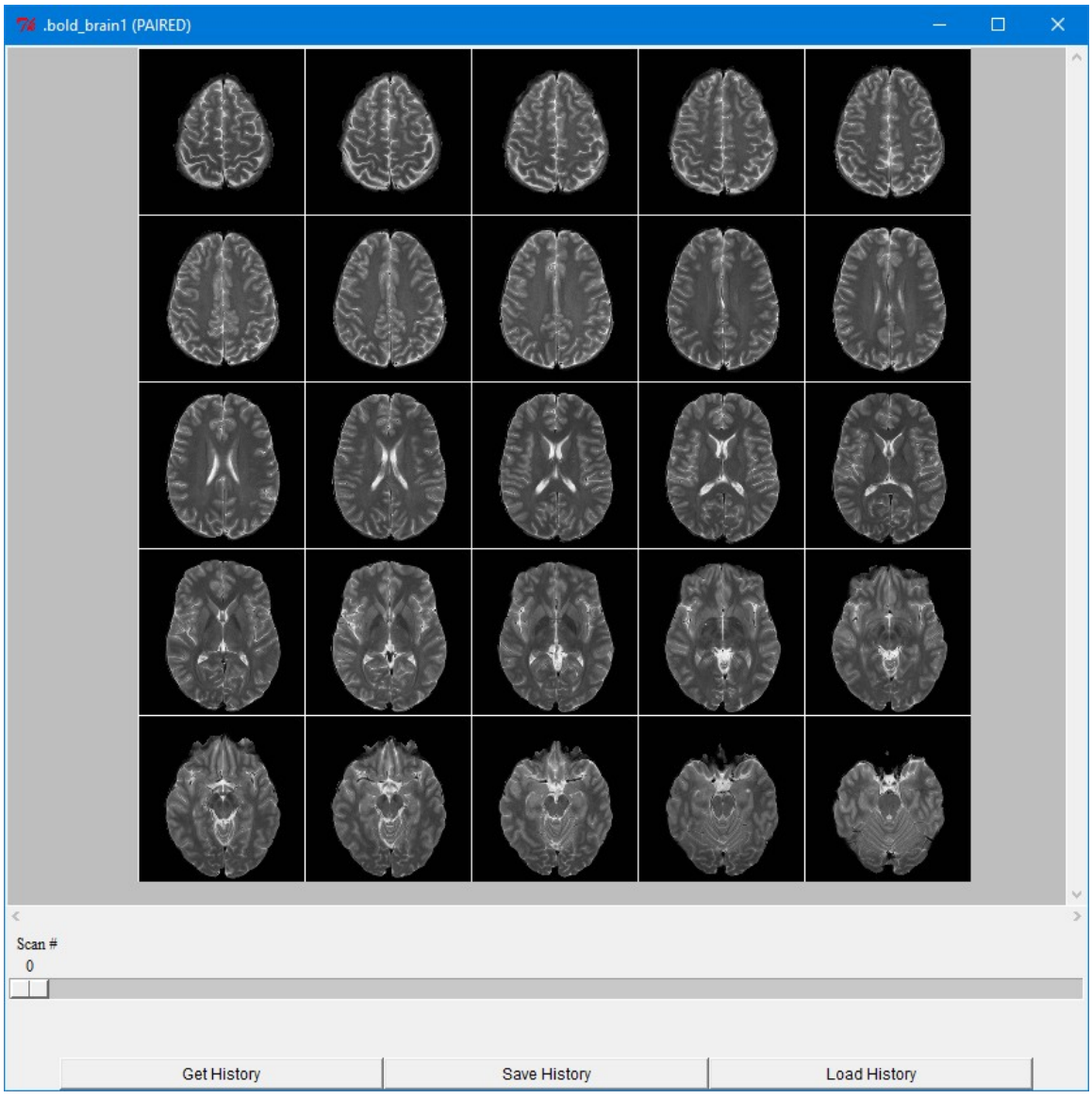


Selecting “Scale all” will scale all the data equally based on the buffer with the highest value being mapped to 1.0:



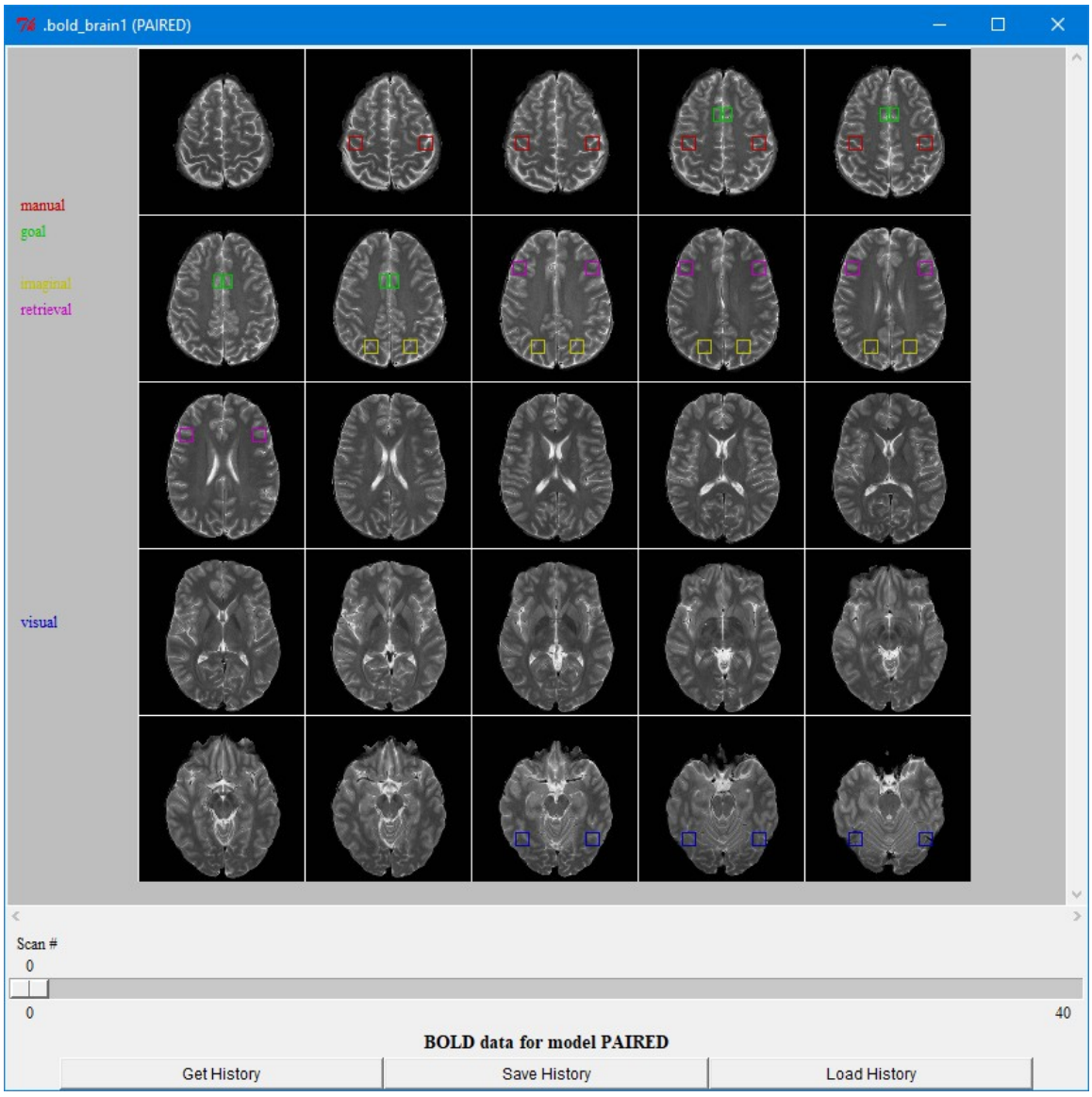
2D brain

The “2D brain” display will show the activity of the buffers overlaid on images of fMRI brain slices. Here is how it looks when first opened:



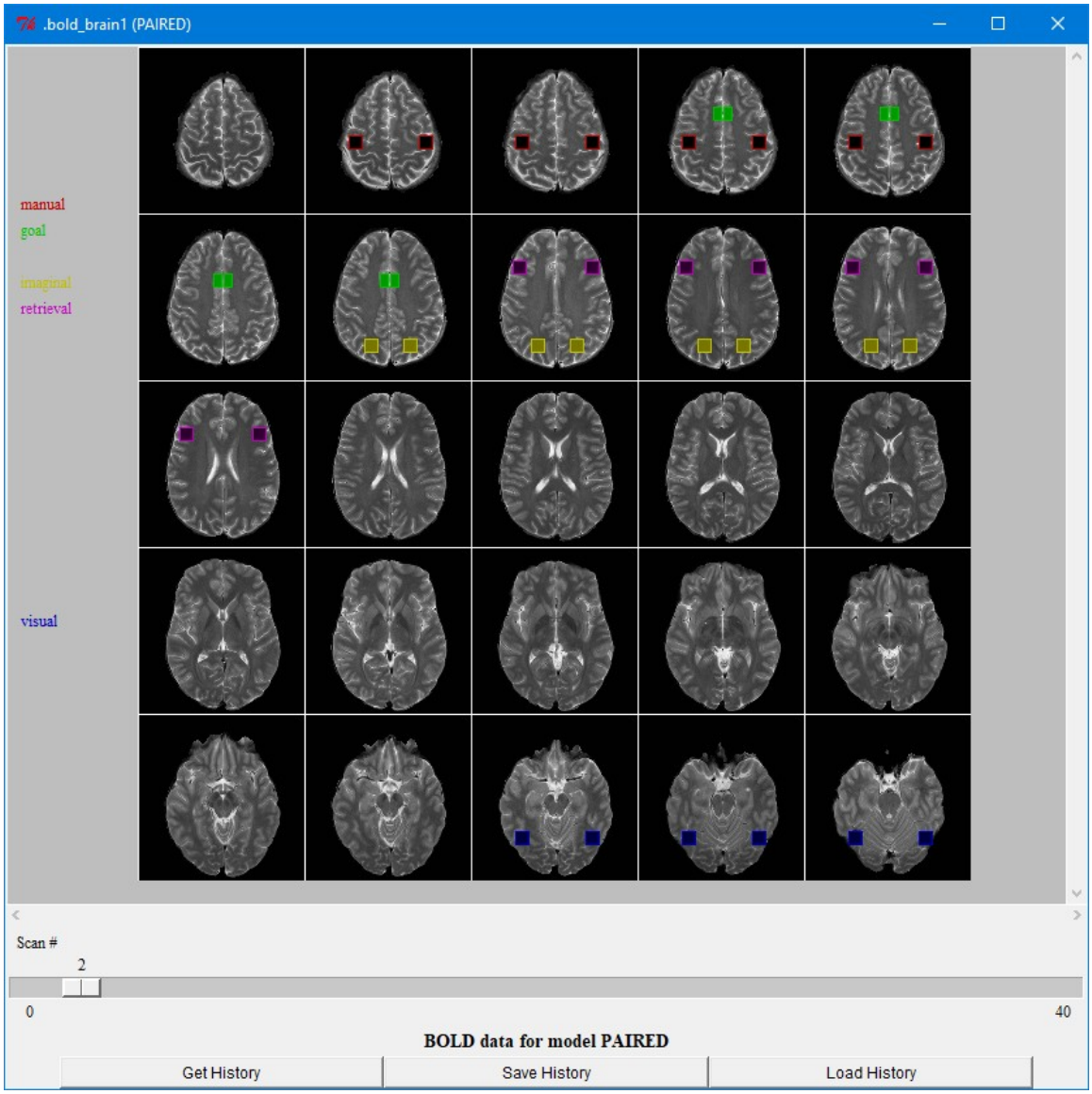
When you get the data it will display the buffers which have data recorded on the left and corresponding colored boxes on the brain images to show the regions associated with that

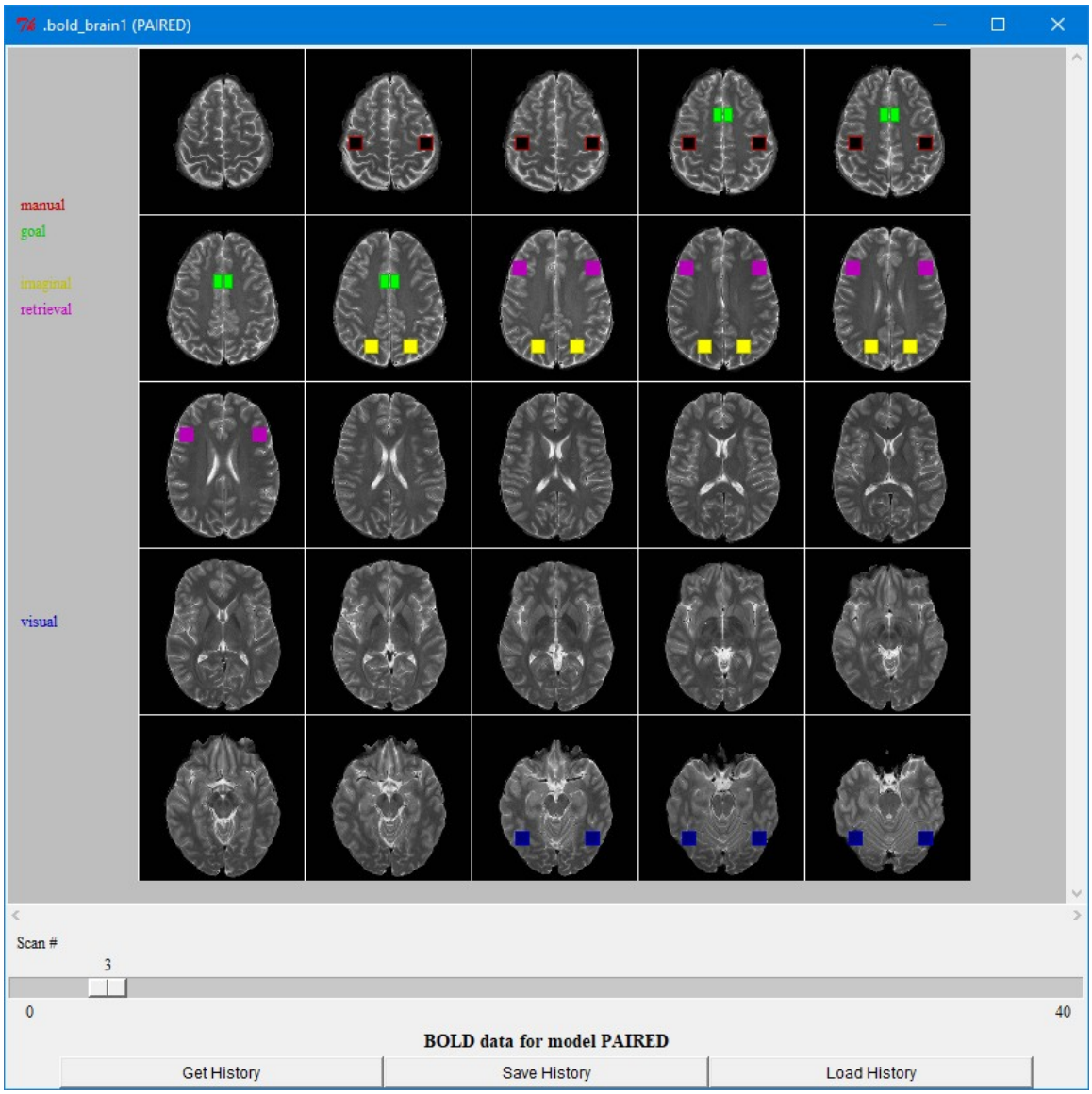
buffer.

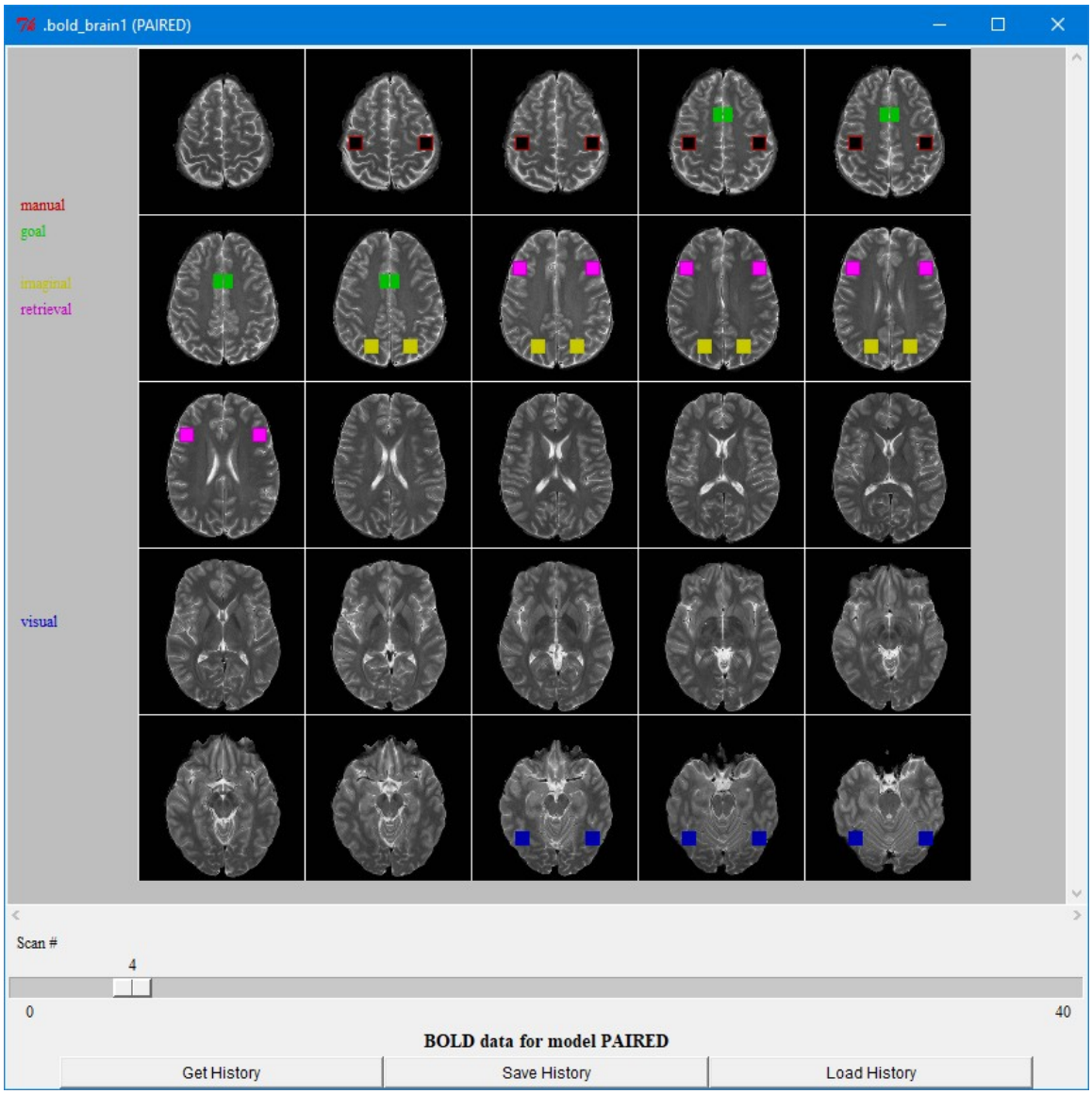


The slider along the bottom allows one to select the specific scan from the run for which the data should be displayed. The scans occur based on the value of the **:bold-inc** parameter, with a scan occurring every **:bold-inc** seconds. On each scan the brightness of the corresponding boxes indicates the BOLD activity in that buffer. Each buffer has its BOLD data scaled from 0.0-1.0 individually and that is used as a brightness value in displaying the color. Thus, if there is no activity, a value of 0, then the box will be black

and if there is a lot of activity, a value near 1.0, then the box will be brightly colored (note however that there is no data for scan 0 and thus the brain image still shows through the box). Here are images from the paired associate model from unit 4 on scans 2-4 showing activity in several buffers increasing at the start of the task:

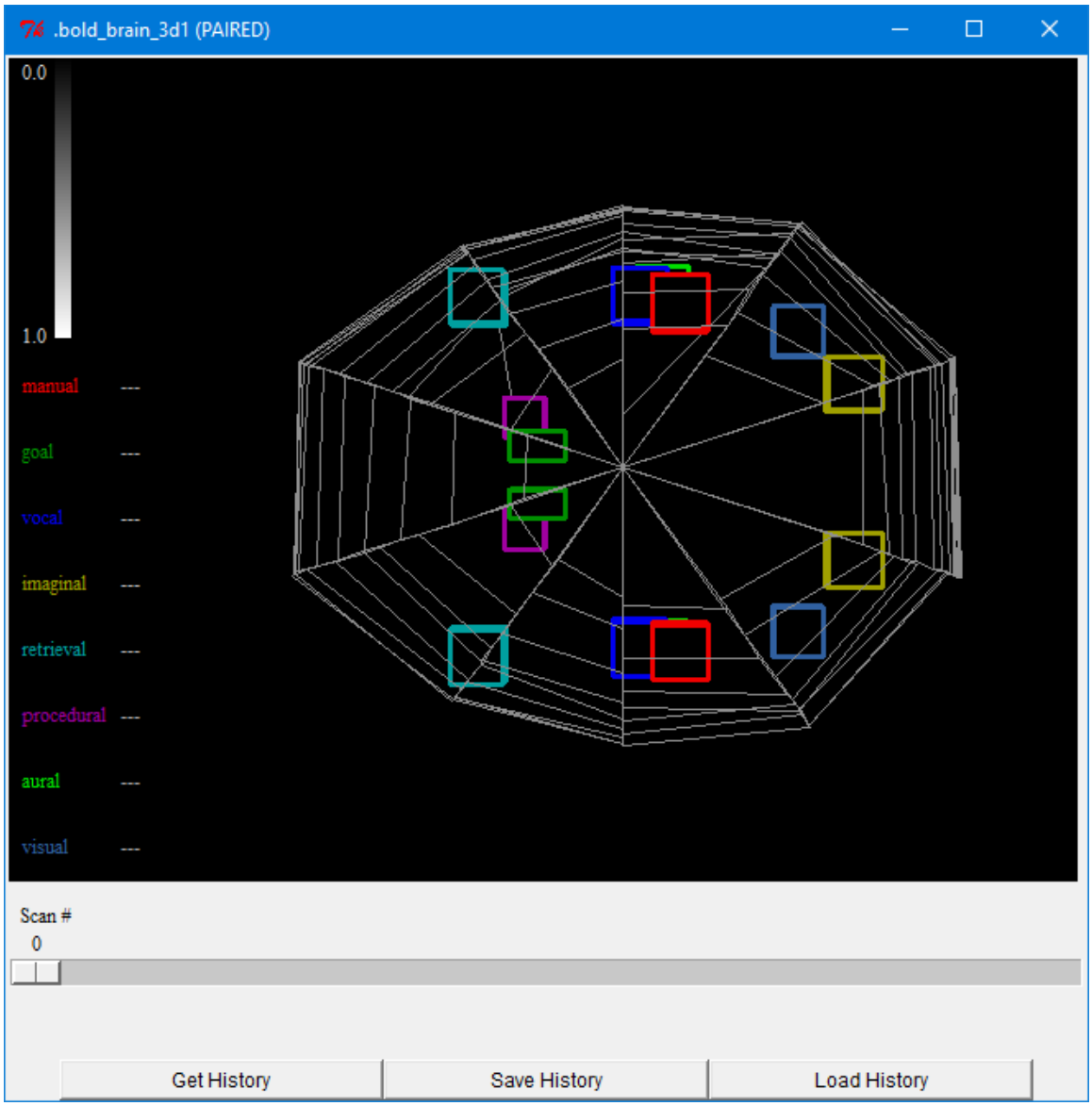



.bold_brain1 (PAIRED)
manual
goal
imaginal
retrieval
visual
Scan #
3
0
40
BOLD data for model PAIRED
Get History
Save History
Load History



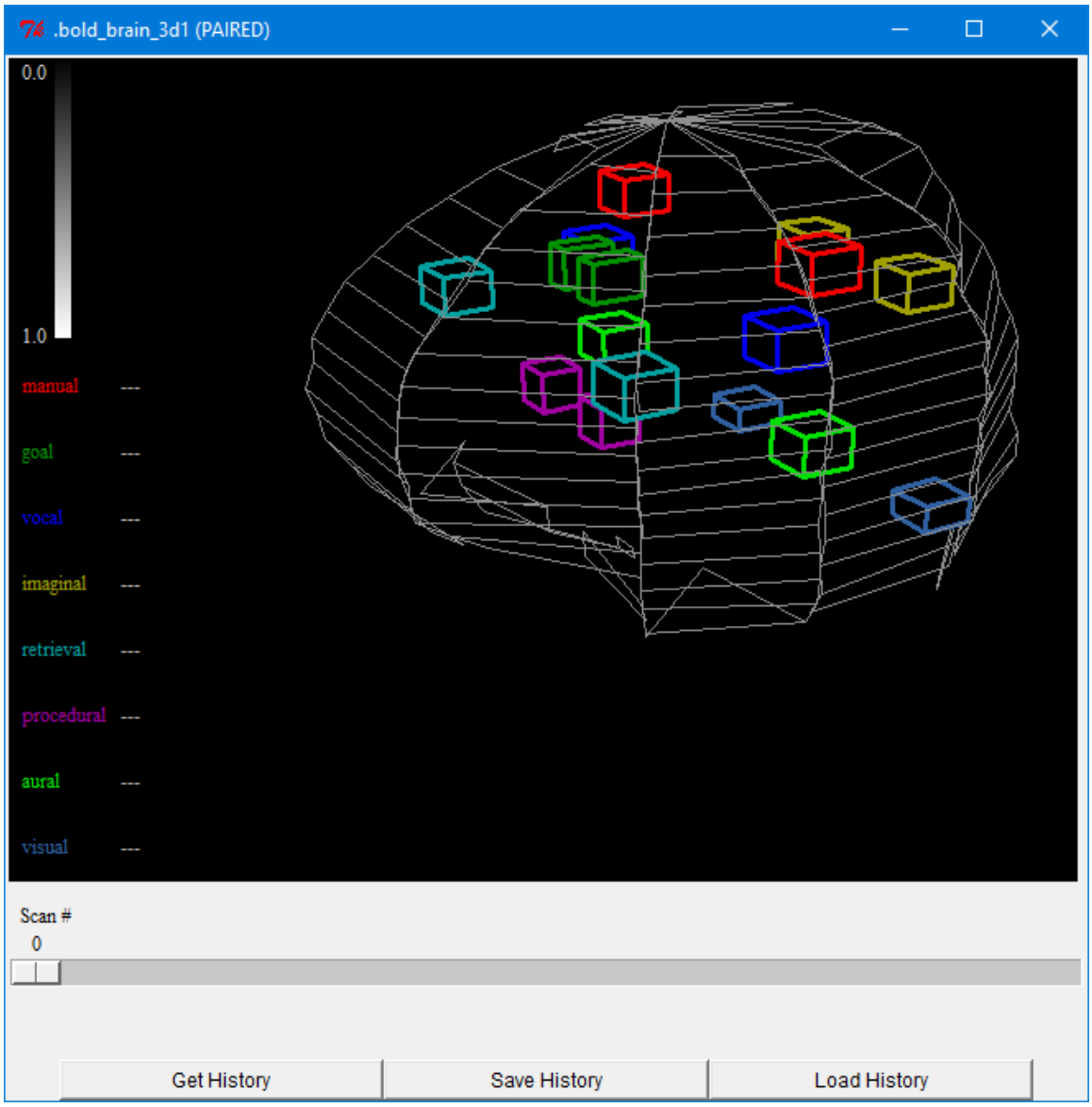
3D brain

The “3D brain” view shows the same information as described above for the 2D brain, except that instead of the using images from a reference brain the boxes are drawn in a very crude three-dimensional wireframe brain model. Here is what the window looks like by default:



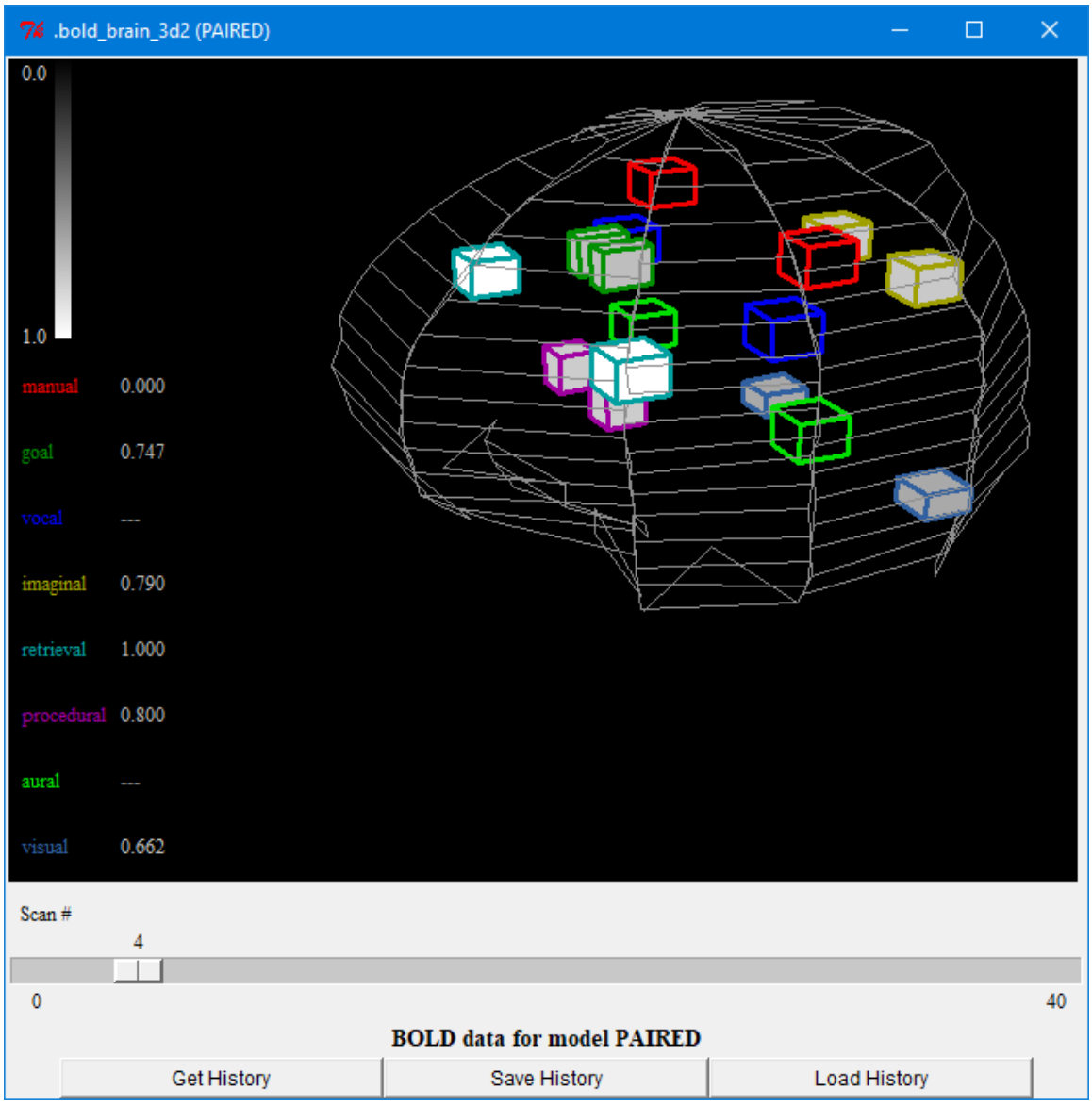
The buffers for which a brain association is defined are displayed on the left in the color which is used to draw the outline of the region's box in the image. The default view is top-down with the front of the brain to the left, but the brain can be rotated and moved by clicking on it and moving the mouse. If the left mouse button is clicked and held moving the mouse will rotate the brain around its center point. If the right mouse button is clicked and held moving the mouse up and down will zoom in and out on the image, and if the middle mouse button is clicked and held moving the mouse will move the brain around in the window without rotating it. Here is a view of the image after it has been moved and

rotated:



The slider along the bottom allows one to select the specific scan from the run for which the data should be displayed in the same way that it does for the 2D viewer. On each scan the boxes for each buffer will be filled with a gray-scale color which indicates the BOLD activity in that buffer at that time. There is a reference scale of the gradient shown in the upper left of the window. The box outlines will always be drawn with the brightly colored edges. Each buffer has its BOLD data scaled from 0.0-1.0 individually and that is used as a brightness value in displaying the color and that number is also shown on the left of the

window after the buffer name on each scan. Here is an image from the paired associate model as run for the graphing data on scan 4 showing activity in several buffers:

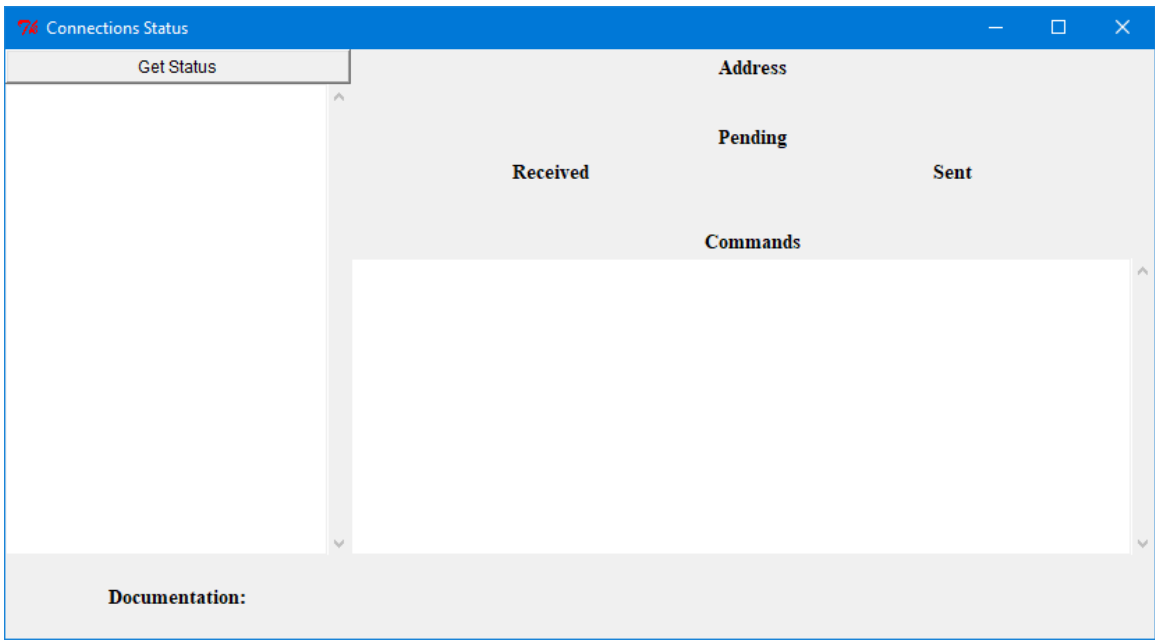


Miscellaneous

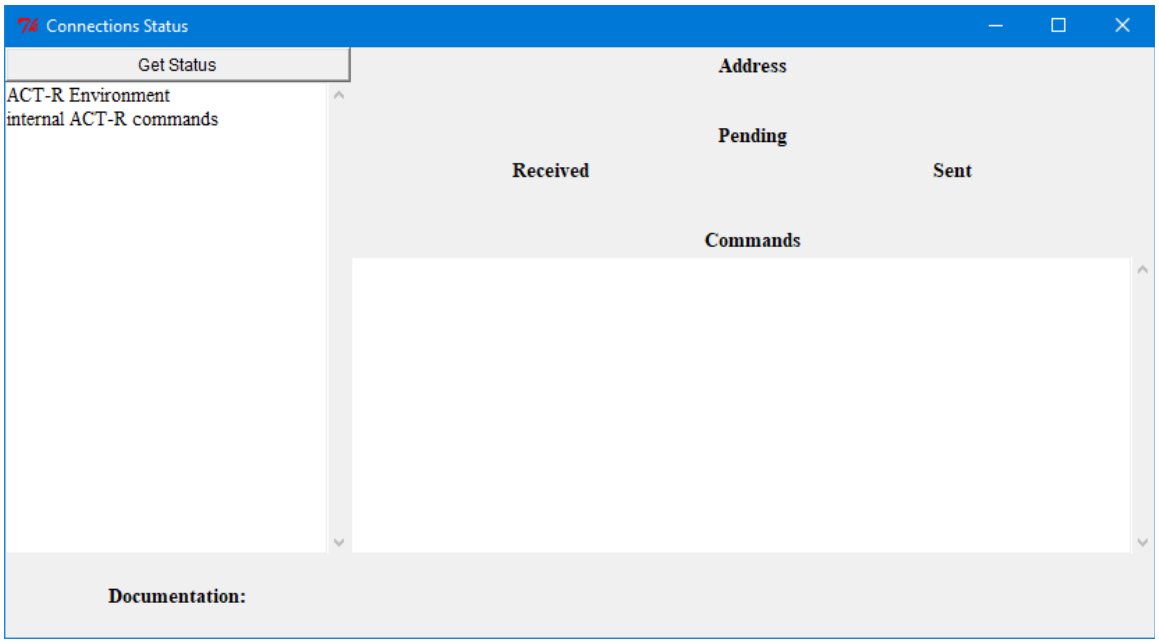
The Miscellaneous section contains the controls which are not involved with the actually modeling and thus do not belong to one of the other sections.

Connections & Commands

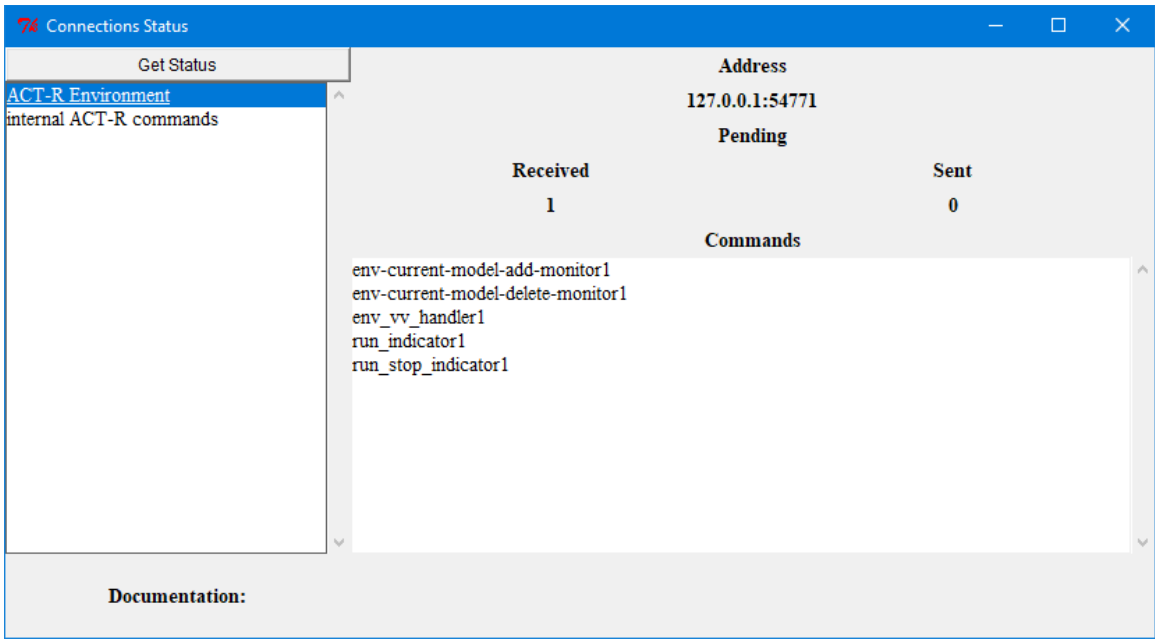
This button will open the "Connections Status" window if it is not already open or bring it to the front if it is open because there is only one such window open at a time. Here is what that will look like:



This window will show the information provided by the ACT-R dispatcher for all of the clients currently connected to it. To get the information you must press the "Get Status" button. That will then add an entry for each connection to the list box on the left. Here is what that will look like if the ACT-R Environment is the only client currently connected to ACT-R:

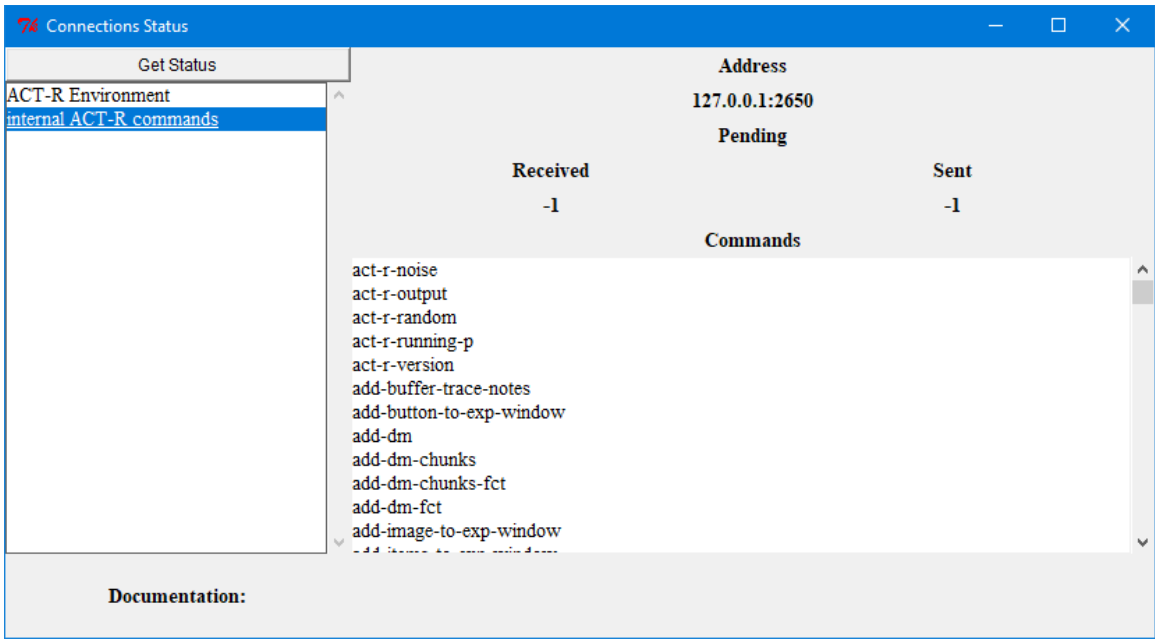


Selecting one of the items from the list will then fill in the information on the right with the information reported for that item at the time "Get Status" was pressed. The Address value will show the ip-address of the client. Pending shows the count of actions which have been Received by ACT-R from the client and which have not been completed and the count of evaluation actions ACT-R has Sent to the client and not yet received a response. Commands will show a list of all the commands which the client has added. Here is what that looks like for the ACT-R Environment:

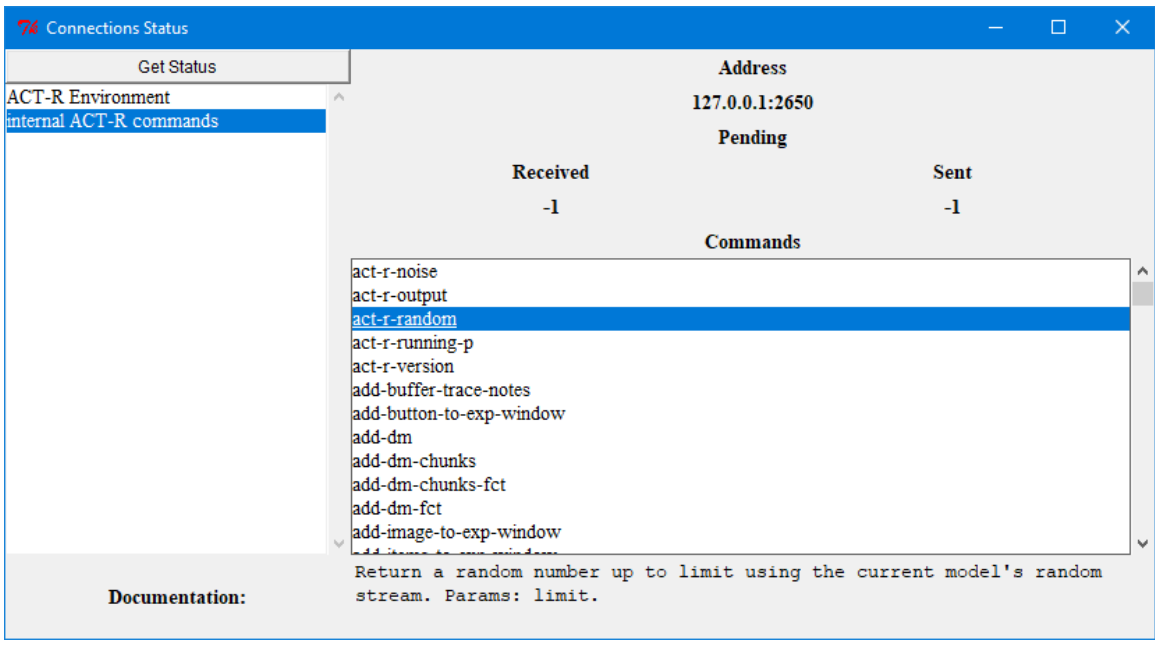


Note that for the ACT-R Environment the Received count will always be at least 1 because when it counts the actions it has not yet completed the request for the status information that it is going to return.

The information for the internal ACT-R commands will show the current ip-address to which clients can connect and the received and sent counts will always be -1 because it does not track the progress of internal actions:

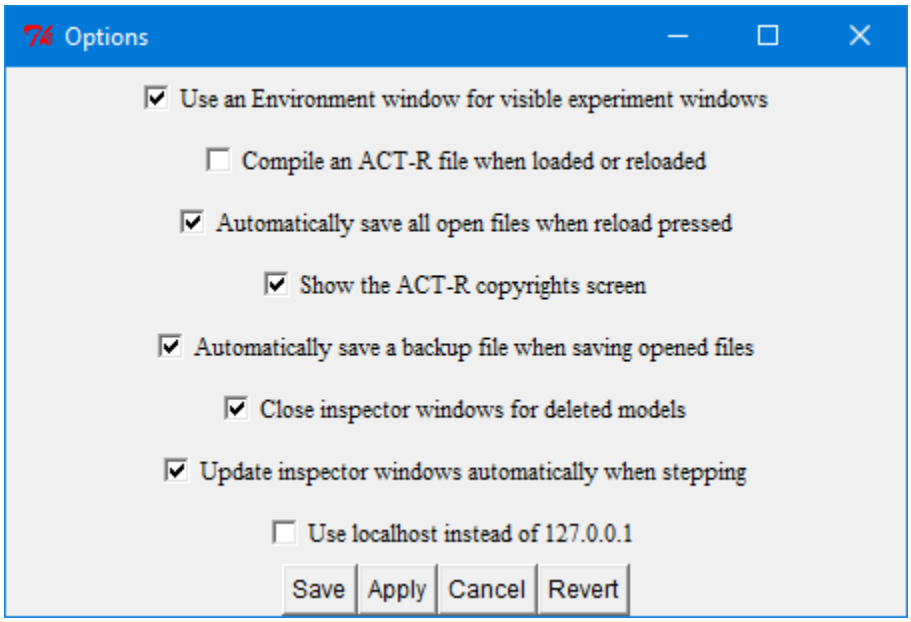


Selecting a command from the list will show the documentation string for that command (if it has one) at the bottom of the window:



Options

This button will bring up the “Options” window if it is not already open or bring it to the front if it is open because there can only be one such window open at a time. Here is what the “Options” window looks like:



It has several options which can be enabled or disabled by checking or unchecking them. When the window is opened the current setting of the options will be shown in the selections. First, the buttons at the bottom of the window will be described, and then each of the options themselves.

Save

Pressing the “Save” button will apply the current selections to the Environment as well as save them to a file so that the next time the Environment is started those settings will be used instead of the defaults. It will also close the “Options” window after saving the settings.

Apply

Pressing the “Apply” button will apply the current selections to the Environment to change how it operates. It does not save those setting for future use nor does it close the

window.

Cancel

Pressing the “Cancel” button will close the window without applying or saving any of the changes which have been made to the options since the last save or apply occurred.

Revert

Pressing the “Revert” button will return all of the options to the values which they had when last applied or saved, undoing any changes that have been made. It does not close the window, nor does it save or apply the values.

Use an Environment window for visible experiment windows

This option indicates whether the Environment should display a visible virtual window for the interfaces created through the ACT-R AGI commands. The default for this option is enabled which means that when the Environment is connected to ACT-R any experiment window opened by the AGI which is marked as being visible will be displayed in a window opened by the Environment. If this option is not enabled then the Environment will not display the virtual windows, but they may still be displayed by other interfaces connected to the AGI for that purpose.

Compile an ACT-R file when loaded or reloaded

This option controls how files are loaded or reloaded through the Environment. If the box is unchecked (the default) then files which are loaded with the “Load ACT-R code” button are loaded directly with the Lisp command load, and when the “Reload” button is pressed the ACT-R **reload** command is called with no parameters.

If the option is enabled then when a file is loaded or opened the file is first compiled with the Lisp command compile-file and then that compiled file is loaded with the load command. When the “Reload” button is pressed with this option enabled the value of **t** is

provided as the optional parameter to the **reload** command which will cause it to also compile the file before loading. This option can be useful if one is using a Lisp which does not compile code automatically, but is not necessary when working with the standalone version of ACT-R since CCL (Clozure Common Lisp, the Lisp used to build the standalone) does compile functions automatically.

Automatically save all open files when reload pressed

This option affects the operation of the Environment when using the "Open File" button to edit files. If this option is enabled, which is the default, then any changes made to the currently open files will be saved before the **reload** command is called in response to pressing the Reload button. If this option is not checked then the changes will not be saved by the Environment prior to the **reload** command being called. That means if one is editing the file of the current model the previously saved version will be loaded.

Show the ACT-R copyrights screen

This option controls whether or not the window showing the ACT-R copyright information is displayed every time the Environment is started. If it is enabled then the window will be displayed and if it is disabled then it will not be displayed.

Automatically save a backup file when saving opened files

This option affects the operation of the Environment when using the "Open File" button. If this option is enabled, then whenever a file is saved (including the automatic saving upon **reload** if enabled) a backup copy of the existing file is made first. The backup copy will have the same name as the original file with an increasing number added to the end of the file's extension. It will be written to the same directory as the original file. Thus, if the count.lisp file were opened and then reloaded a file named count.lisp-0 would be created in the same directory as the original count.lisp file and would be a copy of the file count.lisp before any current changes are saved into it. If it were reloaded again then a file named count.lisp-1 would be created, and so on.

These backup files are intended only as protection against a crash of the system or other error which may cause the loss of the file being worked on. ***The original file will always be the most recently saved version of the model and you should not open or load the backup files directly unless absolutely needed.*** After the Environment has been successfully closed you should feel free to delete any and all of the backup files. If the system does crash or for some other reason you would like to use one of the backup files you should first rename it to something meaningful if you plan to open it in the Environment because otherwise it will also have backups made of it which would then look something like count.lisp-1-0 and that can become confusing very quickly.

Close inspector windows for deleted models

If this is enabled (which is the default setting) then this controls what happens to the inspector windows for models which are no longer defined. If the option is set then when a model is deleted the inspector windows for that model will automatically be closed in the Environment. If it is not set then the windows for deleted models will remain open, but will no longer function and trying to use them may result in warnings from ACT-R.

Update inspector windows automatically when stepping

If this is enabled (which is the default setting) then when the Stepper is being used, all open tools from the Current Data section of the Control Panel will automatically update their contents after the Step or "Run Until:" button is pressed (the updates happen when the "Last Stepped" event updates). If it is not enabled then the tools will not update until selected, which is how they operate when ACT-R is run without the Stepper being open.

Use localhost instead of 127.0.0.1

If this is enabled (the default setting is not enabled), when the Environment tries to connect to ACT-R and the address it finds for the connection is "127.0.0.1" it will use the string "localhost" instead. If you get the dialog that says "Error occurred trying to connect to ACT-R" when the Environment starts and pressing the No button succeeds then enabling this option should avoid that message in the future and connect directly at

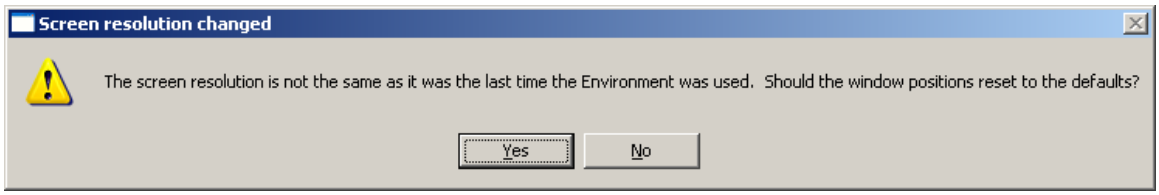
the start.

Additional Environment Settings and Control

Window Positions and Sizes

When the Environment is closed it will save all of the settings for the window sizes and positions that were used while it was running. The next time that the Environment is run it will then use those same sizes and positions for all of the windows.

This is generally desirable, but can lead to problems if one uses the same machine with different displays or monitors which have different resolutions which could result in windows or tools no longer being within the bounds of the current display. To accommodate that there is a test performed when the Environment first starts to check whether or not the same screen space is available by testing the current height and width as well as the maximum window size as reported by the system. If those differ, then it will display this dialog before starting with the option of restoring everything to the default size and location instead of using the saved values:



If the “Yes” option is chosen then the default window positions will be used instead of those in the saved configuration.

If for some reason the Environment doesn’t detect that your screen has changed or you are having some other problems whereby the tools are no longer available because they are opening in windows outside of the screen then you can manually remove the file with the saved settings in it and force things into their default positions the next time the Environment is started. The settings are saved in the Environment’s GUI/init directory in the file named 10-userguisettings.tcl. If you delete that file then the next time the Environment starts all of the windows will revert to their default sizes and positions.

Extending or Changing the Environment

It is possible to add new tools or capabilities to the Environment. Like the ACT-R loader, when the Environment starts it will automatically load source files located in some of its directories. Thus, putting new files into those directories will cause that additional code to be loaded and become a part of the Environment. [Note however that there is currently no documentation or support provided for how the Environment itself operates and thus one is on their own to figure out how to create a new tool.]

When the Environment starts it first loads all of the .tcl files in the GUI/init directory. Then, it opens the Control Panel window and loads all of the .tcl files located in the GUI/dialogs directory. The files are loaded in ascending order based on their file names.

It is also possible to remove tools that you don't need, if you don't want them included in the Environment, by removing those files from the GUI/dialogs directory. Each of the tools in the Environment is generally implemented in a separate file located in the dialogs directory and the names of the files indicate the tool that they implement, for example "38-visicon.tcl" implements the "Visicon" button. Removing that file will remove the "Visicon" button from the Environment. The numbers on the fronts of the names are used to ensure that they are loaded and created in a specific order. Most of the files for the provided tools are independent and it should be safe to remove them without affecting the operation of the others. There are a few exceptions however. The buttons which have an options menu to their right add the button in one file and the options in separate files. Removing the button file itself will cause problems if all of the options are also not removed, but a file implementing one of the options can be removed safely.

There are some additional controls for the Environment included with ACT-R in the environment/GUI/dialogs directory. The files with a .tcx extension can be renamed with a .tcl extension to add them to the Control Panel. Most are simply additional buttons which provide a shortcut to functionality available already e.g. a separate button to bring up a buffer viewer showing the status information. One however, 35a-chunk-tree.tcx, provides

a new chunk viewer and the details on how to use that can be found in the extras/chunk-tree-viewer directory.

Extending the visible virtual windows

It is possible to extend the set of items which the virtual windows of the AGI make available, and to go along with that one can also extend the Environment to display new AGI items which are created. However, there is currently no documentation on how to do so.

Network Configuration Information

To determine how to connect to ACT-R the Environment reads two files which ACT-R writes to the users home directory: act-r-address.txt and act-r-port-num.txt which contain the IP address and socket port number respectively. If those files are not available, then it uses the values which are found in the GUI/init/00-net-config.tcl file included with the Environment.

If you want to connect the Environment to ACT-R running on a different machine then you will need to either edit/create those act-r-*.txt files with the appropriate information for the machine running ACT-R or delete those files and edit the 00-net-config.tcl file to add the appropriate information.

If there are multiple instances of ACT-R running on the same machine, each will have a different port number, but the act-r-port-num.txt file will only contain the port of the last one which was started. If you want to connect the Environment to one of the other instances of ACT-R which is running you will need to edit the act-r-port.txt or 00-net-config.tcl file to indicate the appropriate port (which is printed by ACT-R when it starts).