

ACT-R Environment Manual

Dan Bothell
db30@andrew.cmu.edu

Introduction

This is the current ACT-R Environment. It still has some minor bugs and oddities so any and all suggestions, questions or improvements are welcome. The known problems are listed at the end. It is a radical departure from the old Environments in many ways, but there are many reasons for the change and it is hoped that this Environment will be a more useful tool for a wider audience of old and new ACT-R users alike.

Installation

First, the two pieces of the Environment communicate through a TCP/IP socket, so you need to have TCP functionality on your machine. That typically isn't a problem, but there have been some problems reported when the machine was using PPP for a connection. So, if you are using PPP and experience problems connecting the Environment pieces that may be the reason.

The current version for most systems now includes the Tcl/Tk side of the environment as an executable application, but there are a couple of systems that may not be able to run the application and will still need to have Tcl/Tk installed. I know that MacOS X prior to 10.2 can't run the included application, and there may be some compatibility issues with the Linux version as well.

Unless you are using the standalone version (available for Windows and MacOS 10.2+), you will have to have a supported Lisp application installed. The currently supported Lisps are:

| OS | Lisps |
|-----------|--|
| Windows | ACL free or full version and with or without the IDE for the versions: 6.0 and 6.1 works fully (free or full versions) 6.2 free version only runs interpreted i.e. very slowly 6.2 full version works fully 5.0.1 full version has some bugs listed below 5.0.1 free version requires some minor changes (contact me) 5.0 may work as does 5.0.1, but I don't have it for testing LispWorks 4.2 Personal Edition works (see notes at the end) And the full version should also work (but I don't have a license) |
| MacOS 9.x | MCL 5.0, 4.3.5, 4.3.1 and 4.3 have been tested and work fully, but there are some issues listed below to be aware of |
| MacOS X | OpenMCL works with only a minor issue listed below ACL 6.2 works (it was verified by a user because I don't have a license) MCL 5.0 |

Linux ACL I've only tested 6.1, but other versions should work as for Windows
LispWorks 4.2 also works
CMUCL (I don't know which specific versions)

Other Unix if it has an ACL that should work
CMUCL should also work

If you've got a Lisp that's not currently supported either contact me to see if it's been added or feel free to make the necessary changes yourself (there's some description below as to what's necessary, and it's not that much).

If you don't want to use the included Tcl/Tk application, or you are using Mac OSX prior to 10.2 or Linux/Unix, then you need to make sure that you have Tcl/Tk 8.3.4 or newer installed (actually, it should work with anything since 8.1, but I've been testing in 8.3.4 so that's the safest option). You can get it free from <http://www.tcl.tk/> if you don't have it already, and you should be able to find out just about anything you'd like to know about it there as well. I didn't find it difficult to install and if you have any problems I should be able to help you get it set up.

I assume that you already have the Environment files, but if not you can get them from the ACT-R web site at <http://act-r.psy.cmu.edu/software/>. The one thing to make sure of is that the Lisp source files have the proper line endings for your system (the Tcl files don't really matter because Tcl is pretty lenient when it comes to line endings). If when you load the loader file you get some strange errors that may be a sign that the line endings are "wrong" (if you are using OS X then it actually depends on which Lisp you are using – MCL wants Mac line endings and OpenMCL and ACL want Unix line endings).

Running the Environment

If you are using the standalone version then all you need to do is run the "Start Environment" application, and you do not need the rest of the directions in this section.

Start your Lisp and load the file loader.lisp from the Environment folder. That's going to load the newest ACT-R 5 (which is in the ACT-R folder of the Environment) and the necessary support files to create an environment connection. There's not much in the support files, and until you connect to an environment it should have an almost undetectable impact on the speed of ACT-R (there are a few hook functions set up, but they aren't really doing anything) relative to running without loading them. You're free to work with ACT-R as you typically would at this point (if you don't need the Environment tools yet, don't start them).

You can start the Tcl/Tk side of the Environment now, or wait until you need it. Once it is running you can start and stop the connection to Lisp without stopping the

Environment application if you want. When you're ready to start it the directions depend on which OS you're using:

Windows

Double click the Start Environment.exe application.

Mac OS 9.x

Double click the "Start Environment OS9" Application.

Mac OS X (prior to 10.2)

I don't have an application built for the Tcl/Tk that will run here yet, so you'll have to use the old directions for now. There are basically two ways to get it going, one which uses the terminal and one that doesn't. Without using the terminal, start the wish app (in the Applications folder). Then at the Tcl console window prompt (hit enter if it doesn't give you a prompt right away) enter `cd "/.../GUI"` (where ... is the path to the Environment files) then enter `source starter.tcl`. If you want to use the terminal (or build a script to start it) then you need to `cd` to the Environment/GUI directory and then execute `".../Wish Shell" starter.tcl` (... is the path to the wish shell application and the default installation is at `"/Applications/Wish Shell.app/Contents/MacOS"`).

Mac OS X (10.2 or newer)

Double click the "Start Environment OSX" Application. There is one note to add here. The name of the application is important in this setup, and it will not work correctly if you change it unless you also change the AppMain.tcl script in the included Contents/Resources/Scripts folder accordingly.

Linux/Unix

If you have Tcl/Tk installed already, then you should probably use that with these directions. Change to the Environment/GUI directory and then call wish with starter.tcl i.e. `"wish starter.tcl"`. It's important that the Environment/GUI directory be the working one when it starts, so if you want to make a script to start it automatically keep that in mind.

If you want to use the included Tcl/Tk application then you should run the "startenv" script provided in the Environment directory. If that doesn't work for you, please let me know the details of the error and your machine configuration.

Once you've done the necessary step for your platform it should result in the display of a "Powered by ONR" splash screen (which will go away in three seconds or when you click on it) and then an ACT-R Control panel that says "Waiting for ACT-R *" at the top (with the * being a spinning bar). If you close the control panel while it's waiting it will safely exit the Environment.

When you want to use the Environment tools call **start-environment** from Lisp and that will connect the Lisp to the Environment running on the same machine (it is also possible to connect to an Environment running on a different machine, see below for directions). You should see an ACT-R copyrights dialog for about 5 seconds (or until you click it) if you haven't disabled it in the environment options settings and then you'll get the buttons in the control panel.

When you are done with the environment connection you should call **stop-environment** in Lisp and it will terminate the connection and the Environment application will go back to waiting. If you want to close the Environment application, you can do so now.

That basically covers it for the standard installation and use. Those interested in either the multiple machine Environment setup or the multiple Environments setup can continue with the next section. If you don't need those features you can skip down to the descriptions of the tools available in the Environment.

Running with Multiple Machines

The new Environment has support for two new methods of operation that were not possible in the old Environments. These are still in development, but the basic framework is in place to use it in a limited fashion now. The biggest problem with running the Environment on a different machine at this point is that without access to the listener in the Lisp there isn't a lot that you can do. I've got a listener window working in the standalone version, but it's not quite ready for use on a different machine. So, how much use you'll get out of the split- and multi-Environment features at this point I guess is questionable.

Two machines

The first method is to run the Lisp on one machine and the Environment on a separate machine. Each machine has to have the appropriate application installed (Lisp or Tcl/Tk) and a copy of the Environment. The only difference is the function that you call to make the connection. Instead of calling **start-environment** you need to call **connect-to-environment** and pass the ip address of the machine running the Environment as the host parameter, like this (`connect-to-environment :host "192.168.123.254"`). You should also be able to pass the name of the machine instead of the ip address, like "foo.bar.edu", but I haven't tested that in all of the Lisps so your mileage may vary with that. That's basically all there is to it. You still call **stop-environment** to put the Environment back to the waiting state. Something to note about this is that the machines do not need to be of the

same type or OS – you can connect a Lisp running on a Windows machine to an Environment running on a machine with Mac OS X for instance.

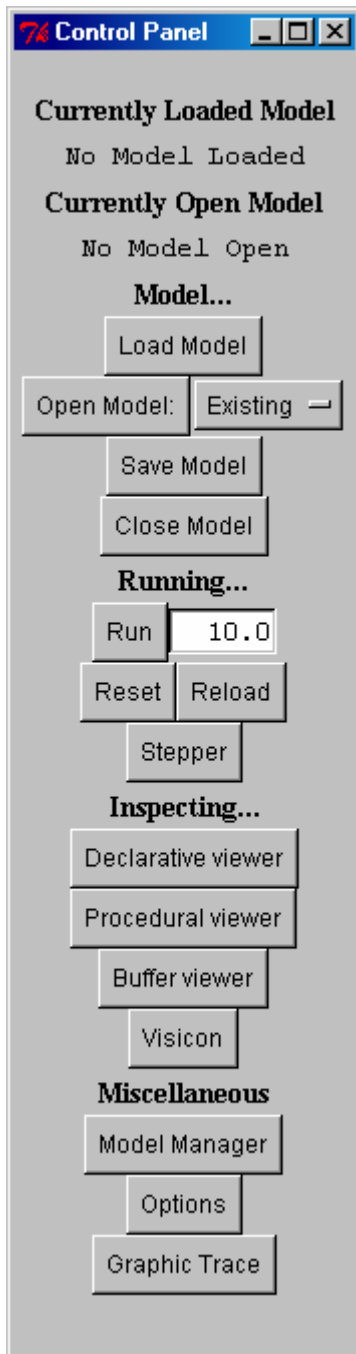
More than two machines

The other new possibility, which is related to the previous one, is to have multiple Environments connected to a single Lisp running ACT-R. Thus, it's possible for multiple people to view/operate a single running model. To connect the first Environment, use either the standard method (if it's on the same machine) or the method described above for a separate machine. Then, to connect the second and subsequent Environments you must call **connect-to-environment** specifying the host address and also the parameter `clean` needs to be `nil` i.e. (`connect-to-environment :host "192.168.123.254" :clean nil`). If you don't specify the `:clean nil` parameter all previously connected Environments will be deactivated (but not completely disconnected until you close them down). To put all of the connected Environments back to waiting you now call **close-all-connections** instead of **stop-environment**, and that will disconnect the Lisp from all of the connected Environments. The big use that I see for this mechanism is remote debugging. Instead of people sending me or Mike error reports with a trace and description of the problem (which often requires several correspondences to resolve) we could schedule a time for that user to connect an Environment to one of us and we'd be able to actually debug the problem on the user's own hardware!

Provided Tools

Many of the current tools work basically the same as the similarly named button or menu item from the old Environments, but there are some differences, and I'll describe each of the items on the Control Panel below. The one big difference that I think makes it worth using is the new stepper. It can step on any of the RPM events and you can pick which ones you want, as well as terminate a run or "fast-forward" to a particular time or production.

On the Control Panel there are a bunch of items, and here's an image of the Windows version (all of the images included are from the Window's version, but the other systems' displays will be structured the same except that they will have the native look for the particular machine):



There are 22 rows of items on the Control Panel. Some of them are only headings to separate sections, but I'll discuss everything there starting from the top and working down. If you are running the standalone version then there will be an additional button labeled Listener between the Options and Graphic Trace buttons which is described in the section on the standalone version.

Currently Loaded Model

At the top is a text heading that says “Currently Loaded Model”. This is a static heading that describes what is on the next line.

Loaded model display

The next row shows a text box that says “No Model Loaded”. This will be changed by the Environment to the name of the currently loaded model file from ACT-R’s perspective automatically whenever a new model is loaded. The model doesn’t have to be loaded from the Environment for this change to occur – it could be loaded in Lisp and this text will be changed to reflect that. Technically, this text shows the name of the file that is the car of the ACT-R global variable `*model*`, or “No Model Loaded” if `*model*` is nil.

Currently Open Model

Next is a text heading that says “Currently Open Model”. This is a static heading that describes what is on the next line.

Open model display

The next row shows a text box that says “No Model Open”. This will be changed by the Environment to the name of the model that is opened in the Environment for editing. This does not have to be the same as the model that ACT-R considers to be currently loaded because one could edit a different model from the one that ACT-R has loaded (though I wouldn’t advise getting into such a situation).

Model...

Below that is a text heading that says “Model...”. This is just a static heading that describes the general functionality of the following items.

Load Model

The “Load Model” button can be used to load a model file into Lisp. When you press it will open a file selection dialog from which you are to select a file to load. That file will then be loaded into the Lisp. If the compile definitions option of the Environment (described below) is enabled, then a separate file will be created that contains all of the definitions (`defun`, `defclass`, `defmethod`, etc) from that file and that file will then be compiled and the resulting compiled file will be loaded as well. Note that this button will only function if the Lisp is running on the same machine as the Environment.

Open Model

The next row contains two items, an “Open Model” button and an option menu. Pressing the button will open a model for editing in the Environment, and load that model into the Lisp as well (as described in the Load Model section). If the split model file option is enabled (as described below) then the model will be edited in the “classic” Environment style of 5 specific windows – Chunk Types, Chunks, Productions, Misc, and Commands. If the split model file option is disabled, then the file will be opened in a single window. There can only be one model opened for editing in the Environment at any time. Note that this button will only function if the Lisp is running on the same machine as the Environment.

The option menu specifies what type of model to open. There are four options. They are Existing (shown as the current selection above), New, Tutor 1.1, and Tutor 1.2. When you press the option menu it will open a list of those choices for you to pick from. If you select Existing, then when you press the open model button you will be presented with a file selection dialog from which to choose a model file. If you select New, when you press the open model button you will be presented with a file selection dialog to specify where to save this model and what to name it, and then an empty model will be created with that name. If you choose either of the tutor options, when you press the open model button you are presented with a file selection dialog to specify where to save this model. Then, the windows for that model opened in tutor mode (note only the windows needed for that model are opened).

While in tutor mode the windows do not operate as normal edit windows. There will be a highlighted selection and when you start typing that selection will be filled in. You should type the value you want followed by either a space or return to have that value accepted. After a correct entry the next part to enter will be presented if there are any more entries to make. At any time you can request help on the current selection by pressing F1. Once all of the required elements have been filled in the tutor will inform you that the model is complete at which time you should save the model, close it, and then open it as an existing model for running.

Save Model

The “Save Model” button will save the contents of the currently opened model file. If there is no model currently opened for editing, then this button has no effect (other than to inform you that there isn’t an open model). When you save a model file it will be checked for (serious) errors in syntax (those that would prevent it from loading into Lisp) and a dialog will be opened to show you if any are found. If the model is open for editing in multiple windows, then the Environment will also check to make sure that everything is in the “right” window, and move things that are not in the right window to where they belong.

Close Model

The “Close Model” button will save the contents of the currently opened model file and close the editing of it i.e. all of its edit windows will be closed, and the open model display will be returned to “No Model Open”.

You should always close an open model before disconnecting the Environment from Lisp. If you do not it will be automatically closed, but it may not be properly “ordered” if it was opened in multiple windows, and you will receive warnings if you do so.

Running...

This is a static text heading that describes the general functionality of the tools to follow. There is also a potential bug with the items in this section, so see the Known Bugs/Issues section below for details.

Run Button

Pressing the “Run” button will execute the pm-run command in the connected Lisp. The editable text box next to the “Run” button allows you to enter the parameter that is passed to pm-run when it is called i.e. the number of seconds to run the model. There can only be one call to pm-run pending at a time from the Environment, so you must wait for the first call to complete before pressing the button again will have any effect.

Reset and Reload Buttons

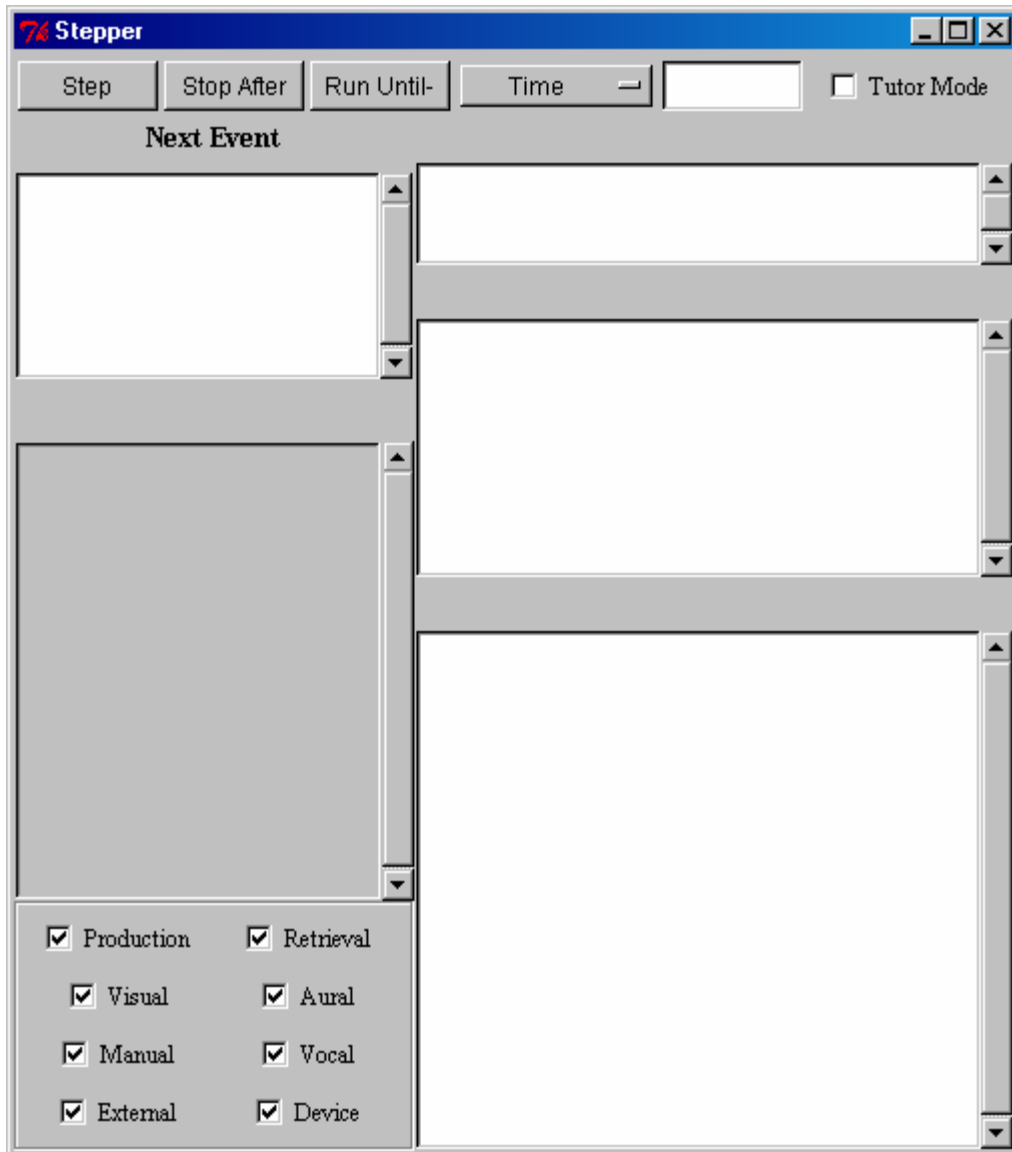
The next row contains two buttons, “Reset” and “Reload”. When these buttons are pressed the corresponding ACT-R function is called in the connected Lisp. In addition, if there is an open model and the automatically save on reload option is enabled the open model file will be saved before the call to reload is made.

Stepper

The “Stepper” button is perhaps the most useful tool in the whole Environment. When it is pressed it will open the stepper dialog 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 stepper dialog open in the Environment (see the Known Bugs/Issues section for information about using the stepper with multiple Environments connected to a single Lisp). The stepper dialog is used to “step” an ACT-R model through its execution one “event” at a time. When it is open it will stop the model at all of the requested points, and wait for you to tell it to continue. To use the stepper dialog you should have it open before you start the model running (if you try to open it while the model is currently running the results are

unpredictable). Thus, the proper way to use it is to open the stepper dialog and then press the “Run” button or call the appropriate function to run the model from a Lisp prompt or the Listener window of the standalone version. If you close the stepper dialog while the model is running the model will continue to run to its natural completion from that point.

The stepper dialog looks like this when you first open it:



This shows the general set of tools it provides and I’ll describe them first. As the model runs other things will be displayed, and I’ll get to those as well.

The Step Button

Pressing the “Step” button makes the model continue to the next stopping event.

The Stop After Button

Pressing the “Stop After” button makes the model execute the currently displayed event, and then the current call to pm-run is terminated. Note that if you are running a model through the use of a Lisp function that contains multiple calls to pm-run, this button only terminates the current call so the model may “continue” to run even after pressing this button.

The Run Until- Button

The “Run Until-” button works in conjunction with the option menu and edit box to its right. When you press this button the model is run without interruption by the stepper until the explicit time or production (as specified by the next two interface items) occurs (or until it reaches the end of the current run) at which point it again stops for inspection. If the current option is time then the model is run until the specified time in seconds occurs. If the current option is production then the model is run until the named production is selected.

The Run Until Option Menu

The option menu has two choices which are time and production (the image above shows time currently selected). When you press the menu it will display a list of those choices for you to pick one. If you pick time, then you must enter a time in seconds in the edit box. If you pick production then the edit box is to contain a production name.

The Run Until Edit box

This box is where you specify the time or production to which the model should be fast-forwarded when the “Run Until-” button is pressed.

The Tutor Mode Checkbox

This checkbox enables tutor mode for the stepper when it is selected. In tutor mode the user is required to fully instantiate all of the productions that are selected before the model will fire them. On a select-production event the bindings will not be shown, and the instantiation will not have the values displayed. Instead, the production will be shown with the variables highlighted. The user is to click on a variable which will open a dialog into which the current binding for that variable is to be entered. Once all of the variables have been correctly bound the stepper can be advanced to the next event. In the dialog for entering the binding there are two buttons called Hint and Help. Pressing the Hint button will provide a suggestion as to how one should find the current value for the variable and the Help button will just show the correct value.

The Next Event Pane

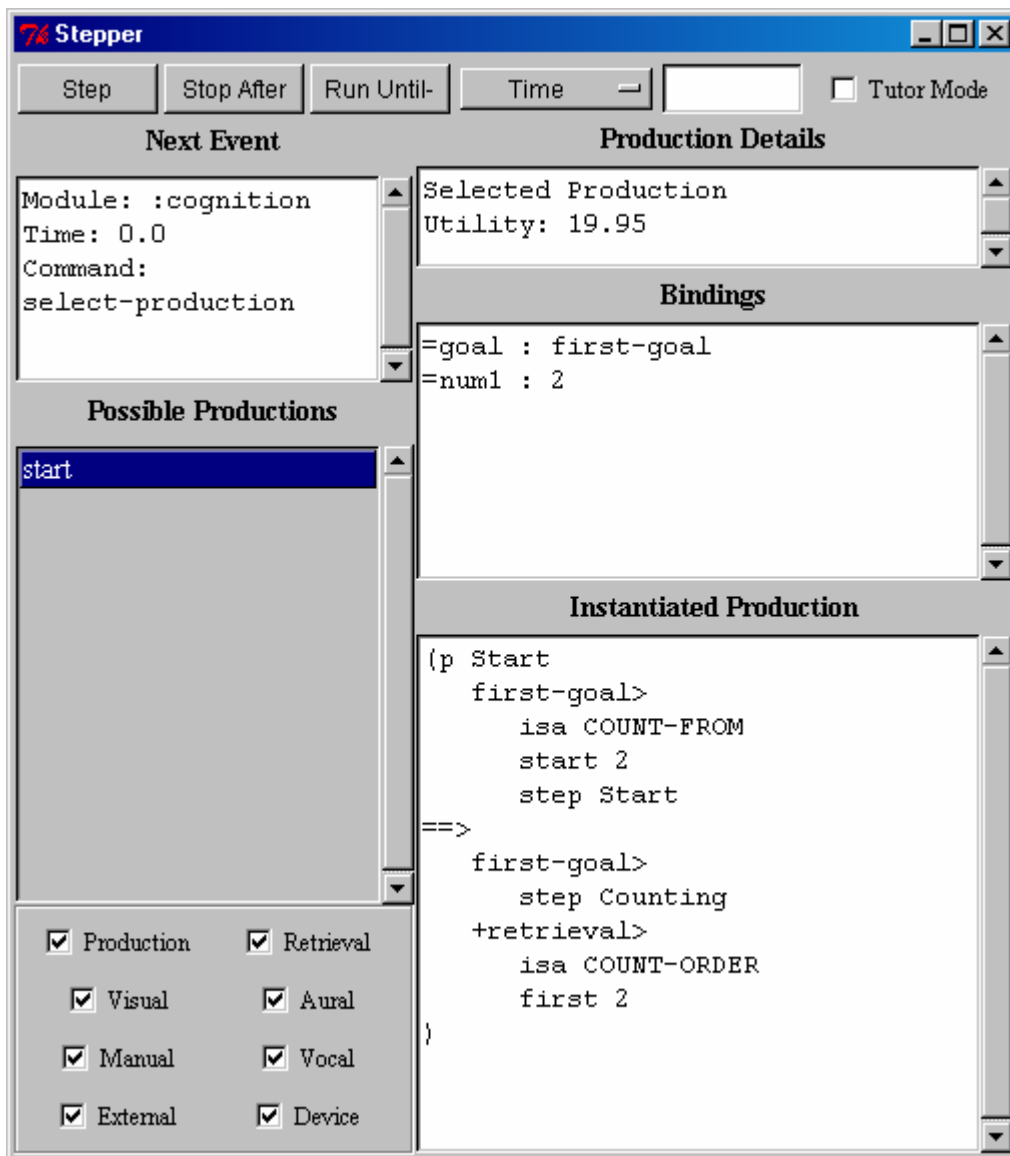
The empty pane in the upper left of the window will be filled in with the details of the event that is going to happen next when the model is being stepped. It will specify which module is to receive, the event, what time the event occurs, and the command that is to be executed.

The Module Selection Pane

In the lower left corner of the window is a box with several check boxes. These are the destinations of events on which the stepper can stop. If a checkbox for an item is on, then the stepper will stop every time one of those events occurs. These can be changed while the stepper is open, so one could turn on a particular module's events only when they were important for instance.

That covers the general options of the stepper dialog. The other panes are important when there is either a production or retrieval event. I'll show an example of each below and describe them.

A Production Event



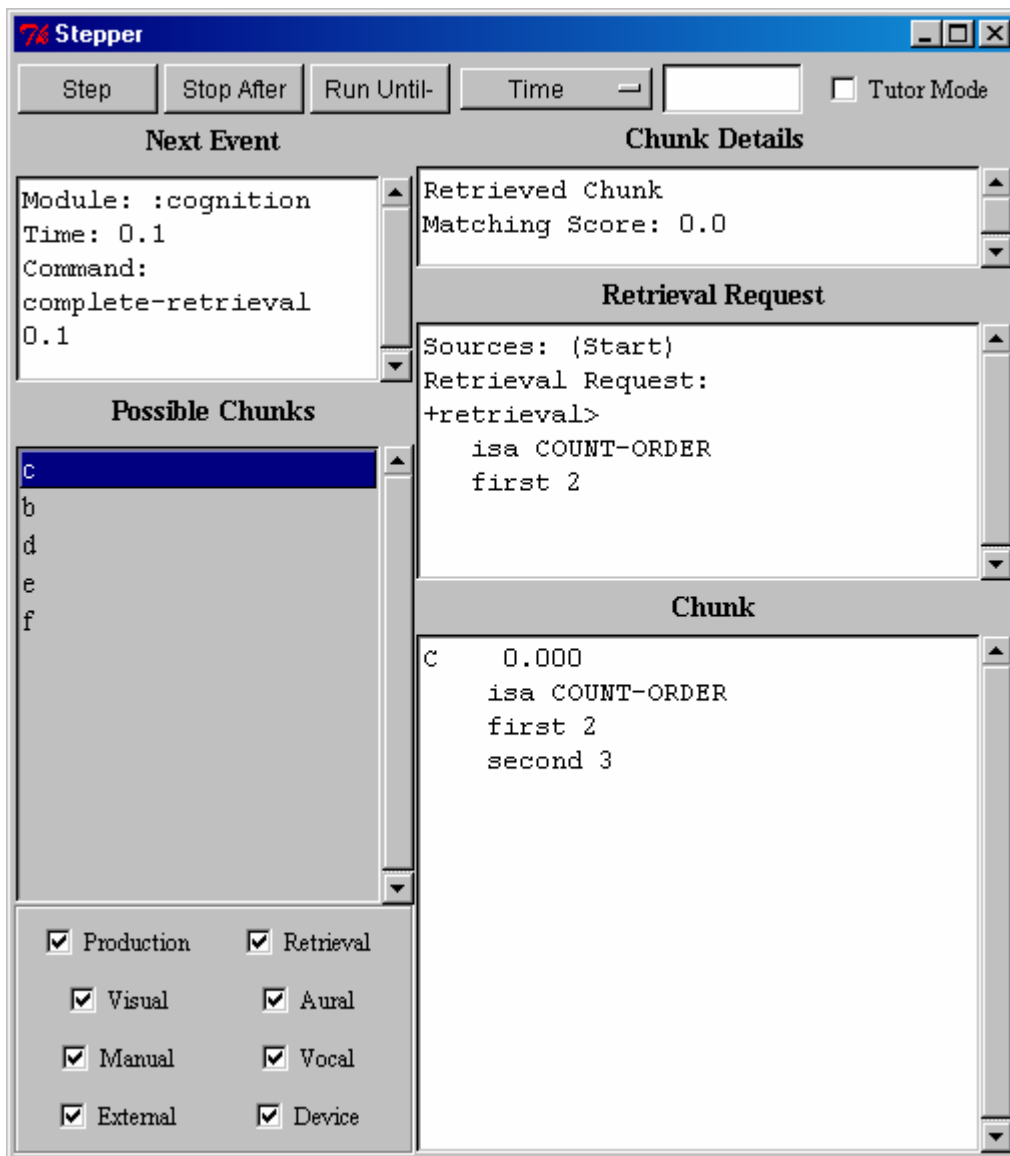
A production event can be generated on either a production selection (select-production command) or a production firing (execute-rhs command). The information displayed is the same in either case. This is basically the same as the functioning of the stepper in the old Environments.

In the middle pane on the left is a list of the productions that were in the conflict set ordered by utility (highest first). The three panes on the right display the information about the production selected from the list. The top pane shows whether this was the chosen production and its utility. The second pane displays the bindings of the variables in that instantiation of the production, and the third pane displays the production instantiated (or at least as much as it could be instantiated).

If you have enabled the “Allow Stepper to pick the instantiation” option, then it is possible to choose which production will be selected next. When that option is enabled the production that is selected in the “Possible Productions” list on the left during a select-production event will be the one selected and fired when the Step button is pressed regardless of its utility (note that it is only the Step button that will force that production to be chosen and not the “Stop After” or “Run Until-” buttons). This functionality is useful for debugging and model tracing when you want a specific action to occur, but don’t want to disable noise and/or utility calculations.

A Retrieval Event

A new feature of the stepper for this Environment is the display of chunks for a retrieval event, shown here:



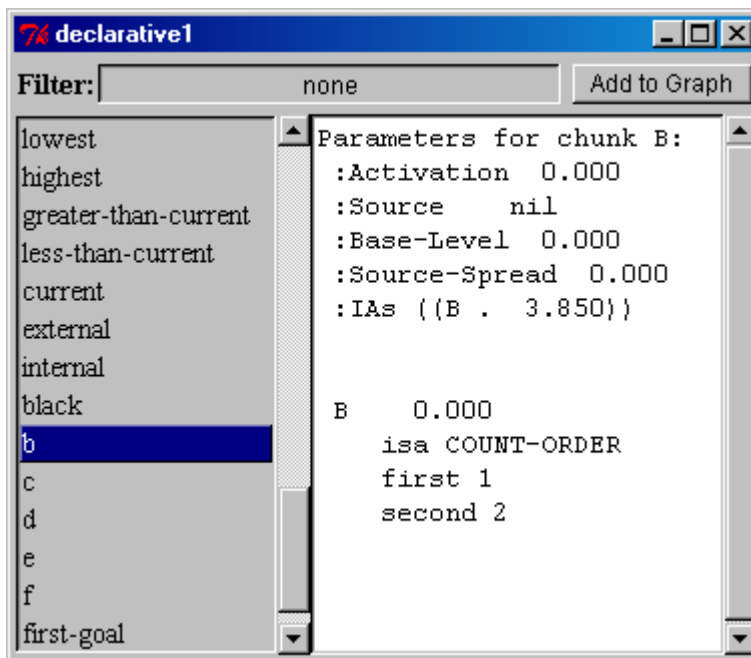
When a chunk is retrieved it will generate an event with the complete-retrieval command. The middle pane on the left shows a list of chunks that were attempted for that retrieval ordered by their matching scores (highest on top). The three panes on the right display the information about the chunk selected from the list on the left. The top pane displays whether it was the chunk that was retrieved and its matching score. If the matching score is nil, then the chunk did not match the specification. The second pane displays the sources of activation that were active when the retrieval request was issued and the retrieval request itself. The third pane shows the chunk that is selected.

Inspecting...

This is a static text heading that describes the general functionality of the tools to follow. All of the inspecting tools are updated as the model runs, however if the model is running at “full speed” (not being stepped through) then you may not see all of the updates as they occur (they may arrive faster than they can be displayed).

Declarative Viewer

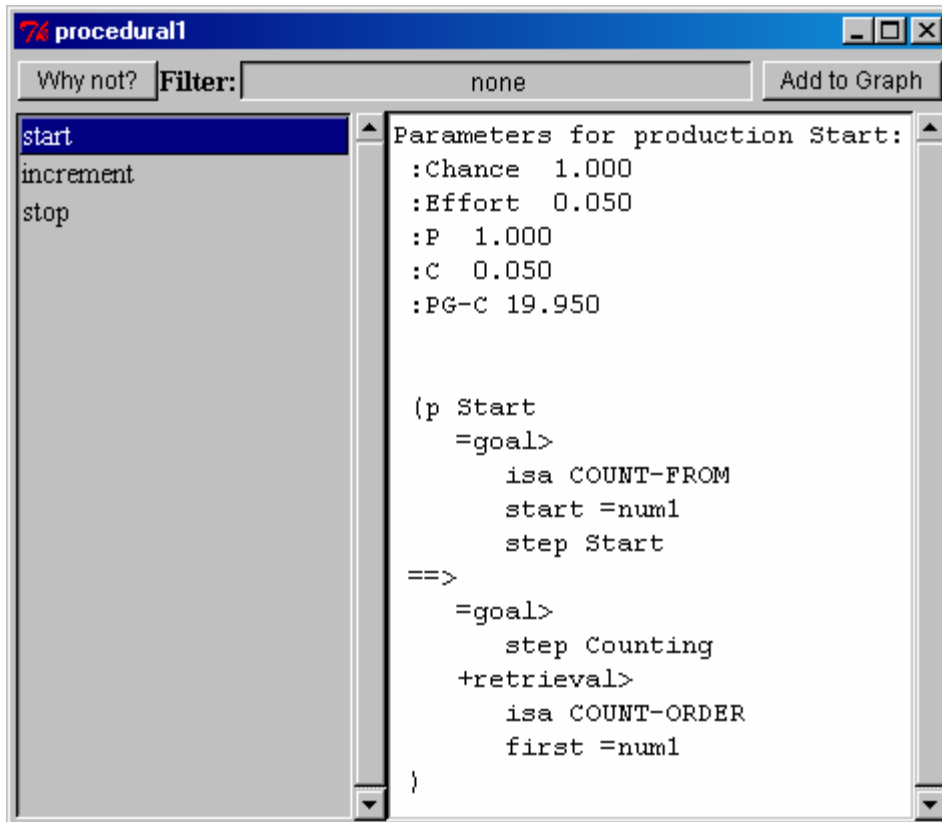
Pressing the “Declarative viewer” button will open a new declarative memory viewer. This differs from the old Environments which only had a single declarative viewer. The declarative viewer will look like this:



On the left side is a list of chunks and on the right is the display of the currently selected chunk from the list. There are two buttons at the top. The one labeled “Add to Graph” does not do anything at this point, but will eventually allow the graphing of parameters during a model run as was possible in the old Windows version of the Environment. The other button (the one labeled “none” above) is the filter. Its name shows which chunk-types are displayed in the list (none means no filter, all chunks are displayed). When you press that button a list of the available chunk-types is shown and if you select one from the list it becomes the current filter and only chunks of that type are displayed.

Procedural Viewer

Pressing the “Procedural viewer” button will open a new procedural memory (productions) viewer. This differs from the old Environments which only had a single procedural viewer. The procedural viewer will look like this:



It operates in a manner similar to the declarative memory viewer. On the left side is a list of productions and on the right is the display of the currently selected production from the list. There are three buttons at the top. The one labeled “Add to Graph” does not do anything at this point, but will eventually allow the graphing of parameters during a model run as was possible in the old Windows version of the Environment. The middle button (the one labeled “none” above) is the filter. Its name shows the current filter for the list of productions. For the procedural viewer the filter is the chunk-type of the goal buffer test on the LHS of the production, or one of the two special filters: *none* which means no filter and all productions are displayed and *no-goal-test* which means only those productions which do not have a goal buffer test on the LHS are displayed. When you press that button a list of the chunk-types used in the goal buffer tests of productions is shown and if you select one from the list it becomes the current filter. The third button is an important debugging tool and is described in the next section.

The “Why not?” Button

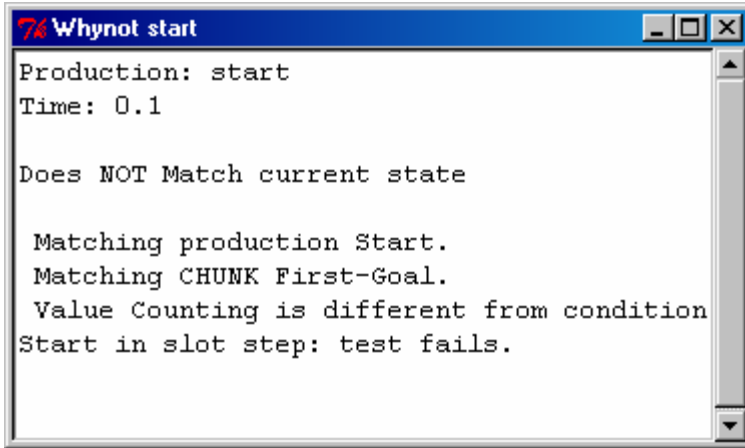
The “Why not?” button is used to call the ACT-R whynot function. That will open a new window which displays whether the current production matches the current buffer contents or not and if so what its instantiation is and if not why it does not match. Here are two whynot windows. The first shows a successfully matching production:

```
74 Whynot increment
Production: increment
Time: 0.1

Does Match current state
Here is the instantiation

Matching production Increment.
Matching CHUNK First-Goal.
Matching CHUNK C.
Instantiation 1:
Increment      nil      19.950
(p Increment
  first-goal>
    isa COUNT-FROM
    start 2
  - end 2
    step Counting
  c>
    isa COUNT-ORDER
    first 2
    second 3
==>
  first-goal>
    start 3
  +retrieval>
    isa COUNT-ORDER
    first 3
  !output! (2)
)
```

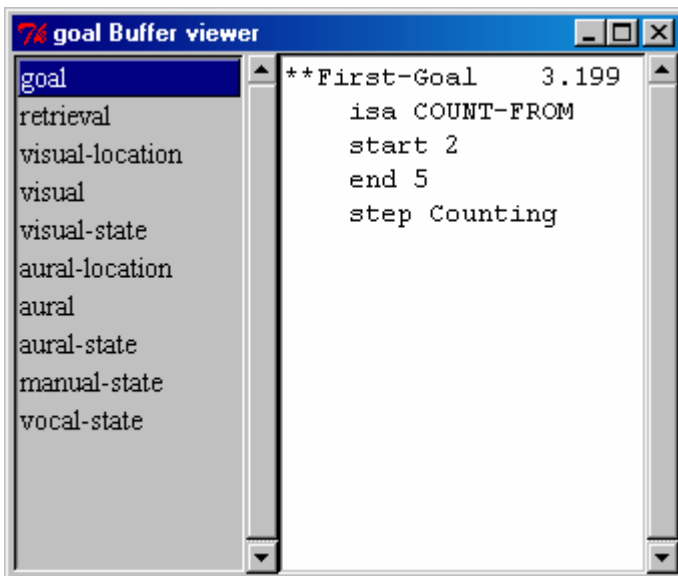
This one shows a production that does not match:



Using whynot in conjunction with the stepper is a very effective method for determining why a model is not doing what you expect.

Buffer Viewer

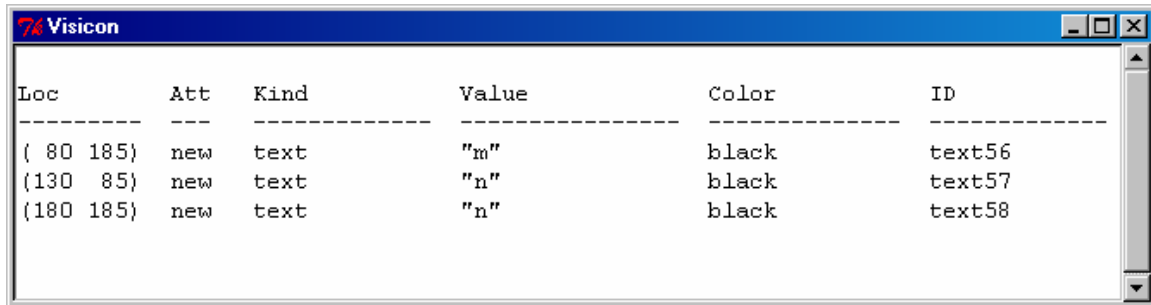
Pressing the “Buffer viewer” button will open a new buffer inspector window. This differs quite a bit from the buffers button in the old Environments. The buffers viewer will look like this:



On the left is a list of the buffers in ACT-R and on the right is displayed the current contents of the selected buffer.

Visicon

Pressing the “Visicon” button displays a window with the current contents of the visual icon (the visicon) if there is not already one open, and brings the current window to the front if there is one already open. It will look something like the following:



The screenshot shows a window titled "7 Visicon" with a table of visual features. The table has six columns: Loc, Att, Kind, Value, Color, and ID. The data is as follows:

| Loc | Att | Kind | Value | Color | ID |
|-----------|-----|------|-------|-------|--------|
| { 80 185} | new | text | "m" | black | text56 |
| {130 85} | new | text | "n" | black | text57 |
| {180 185} | new | text | "n" | black | text58 |

The visicon is the list of features that are currently available for ACT-R to “see”. The display shows the xy location of the feature in pixels, whether or not it has been attended, its kind (type), value (varies based on the type), color and the name of the corresponding visual-object chunk that will be created when/if that feature is attended.

Miscellaneous

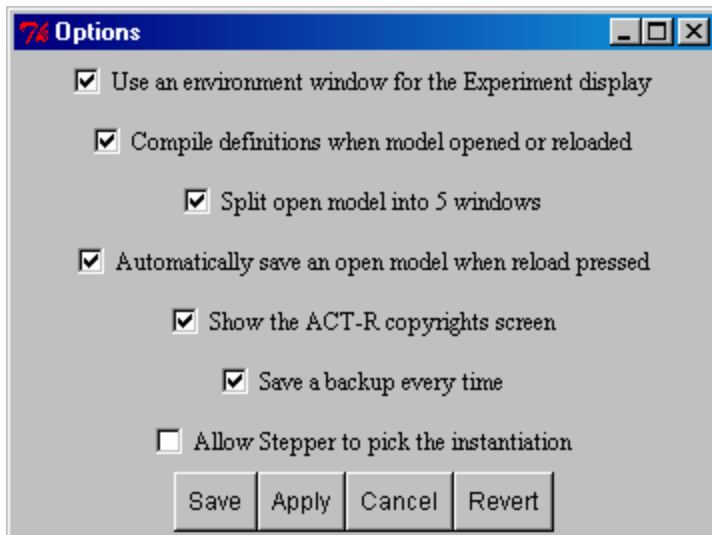
This is a static text heading that describes the general functionality of the tools to follow.

Model Manager

When there is an open model, pressing the “Model Manager” button will open a window with the list of edit windows for the model. Selecting one of the windows in the list will bring it to the front, and if it is not currently open it will be opened again as well.

Options

Pressing the “Options” button will open a window that allows you to configure your Environment (something which the old Environments did not have). The Options window looks like this:



Currently, there are six options available (though one is currently nonfunctional) and four buttons. First I’ll describe the functioning of the buttons.

Save Button

The “Save” button writes the current setting of the options to a configuration file that will be loaded the next time the environment is started and closes the options window.

Apply Button

The “Apply” button causes the current setting of the options to be applied instantly to the Environment.

Cancel Button

The “Cancel” button causes the options window to close and throws away any changes that were made to the options that have not yet been applied or saved.

Revert Button

The “Revert” button resets all of the options to the last applied values, or the values that they had when the options window was first opened if they have not been applied.

Here are the descriptions of what the options do:

Use an Environment Window for the Experiment display: This option sets whether the “visible virtual windows” of the AGI (ACT-R GUI interface whose manual is available from the Tutorial page of the ACT-R website) use a Tcl/Tk window to display the task (when on) or a native Lisp window for the task (when off). Right now, this option has no effect. The system operates as if the option is on regardless of the option setting.

Compile definitions when model opened or reloaded: This option controls whether the definitions in an opened model file are saved out to a separate file which is compiled and loaded (when on) or whether the file is loaded as is (when off). If using a Lisp that doesn't automatically compile definitions, then using this option can have a big impact on the performance of a model if it relies on a lot of Lisp functions for operation i.e. having this on can make the model run much faster in that case. There is an issue listed below that relates to using this option with OpenMCL (which automatically compiles definitions anyway so it should be left off), and when using the standalone version this option must be off.

Split open model into 5 windows: This option controls how a model that is opened is to be edited. If this option is on, then the model is split into the five edit windows as were used by the old Environments, and if it is off a single window is used for editing the model. This option does not affect the tutor models which will always be opened in multiple windows.

Automatically save an open model when reload pressed: If this option is enabled, then if you hit the reload button and there is a model currently open for editing in the Environment, that model will be saved before the reload call is executed in ACT-R.

Show the ACT-R copyright screen: If this option is enabled then every time a connection is made to the Environment from Lisp it will display the ACT-R copyrights for 5 seconds (or until the window is clicked on). If it is disabled, then the copyright screen will never be shown. If you do a lot of starting and stopping of the Environment then you may want to disable this option because sometimes the copyright screen doesn't come to the front, and then you've got to wait 5 seconds before the Control Panel responds each time you reconnect.

Save a backup every time: If this option is enabled then every time the model is saved (either manually or automatically on a reload if enabled) the current version is first copied to a backup file. The backup will have the same name as the original with a #-# appended to the end of it. The # will start at 0 and increase each time. It will not overwrite an existing file with a backup, and will continue to increase the number until a safe name is found (this allows you to maintain backups over multiple sessions). [This option may result in a lot of backup files being generated as you develop a model, but disk space is generally cheaper than a user's time. You should never open one of the backup files directly. If you need to use one you should copy it to a file without a #-# on the end before

opening it. Other wise there will be a backup created of that and you will end up in a confusing situation with files named things like “model-10-4-3-2-0”.]

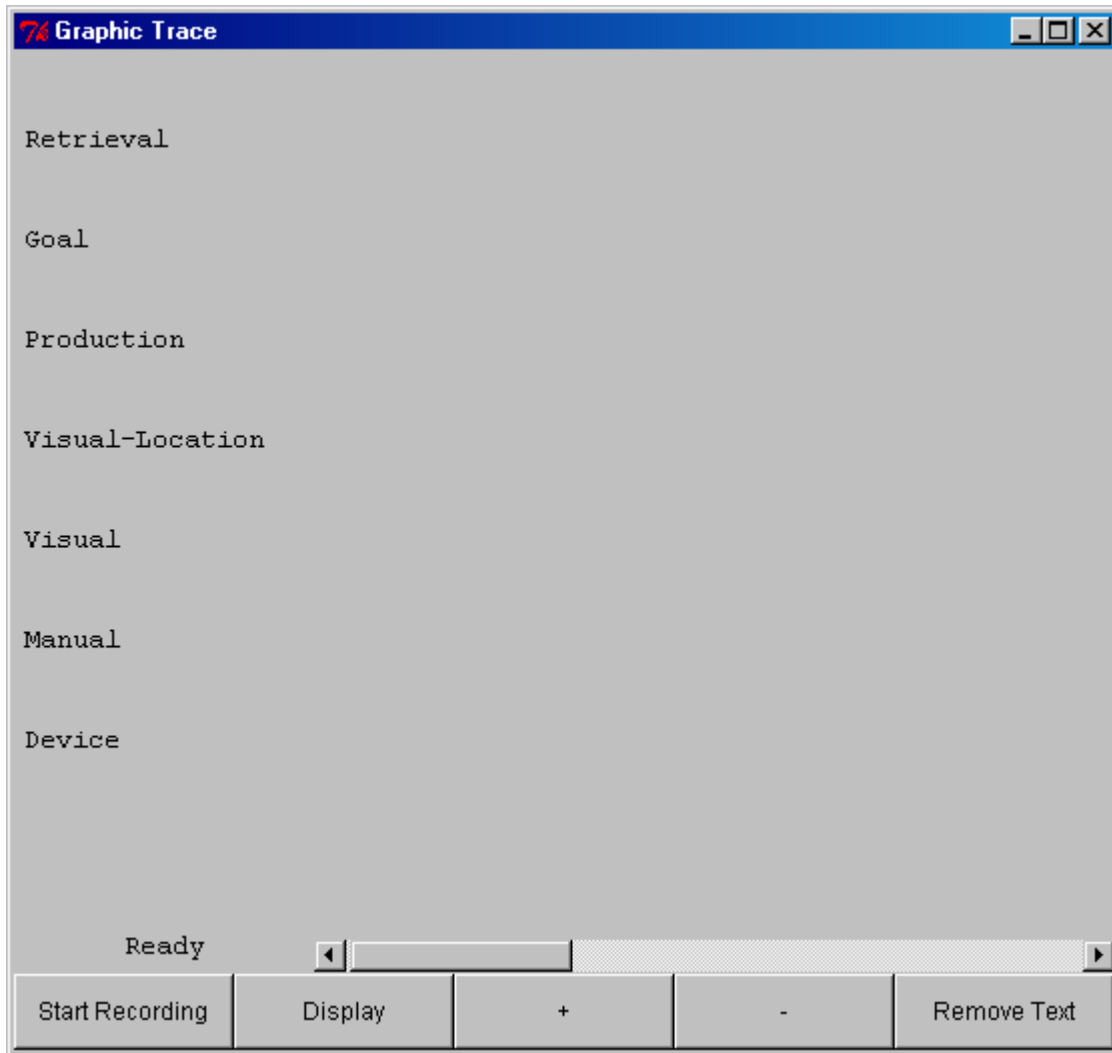
Allow Stepper to pick the instantiation: When this option is enabled it is possible to choose which production will be selected next during a select-production event in the stepper. Details for using this are described above in the Stepper section under the “A Production Event” heading.

Graphic Trace

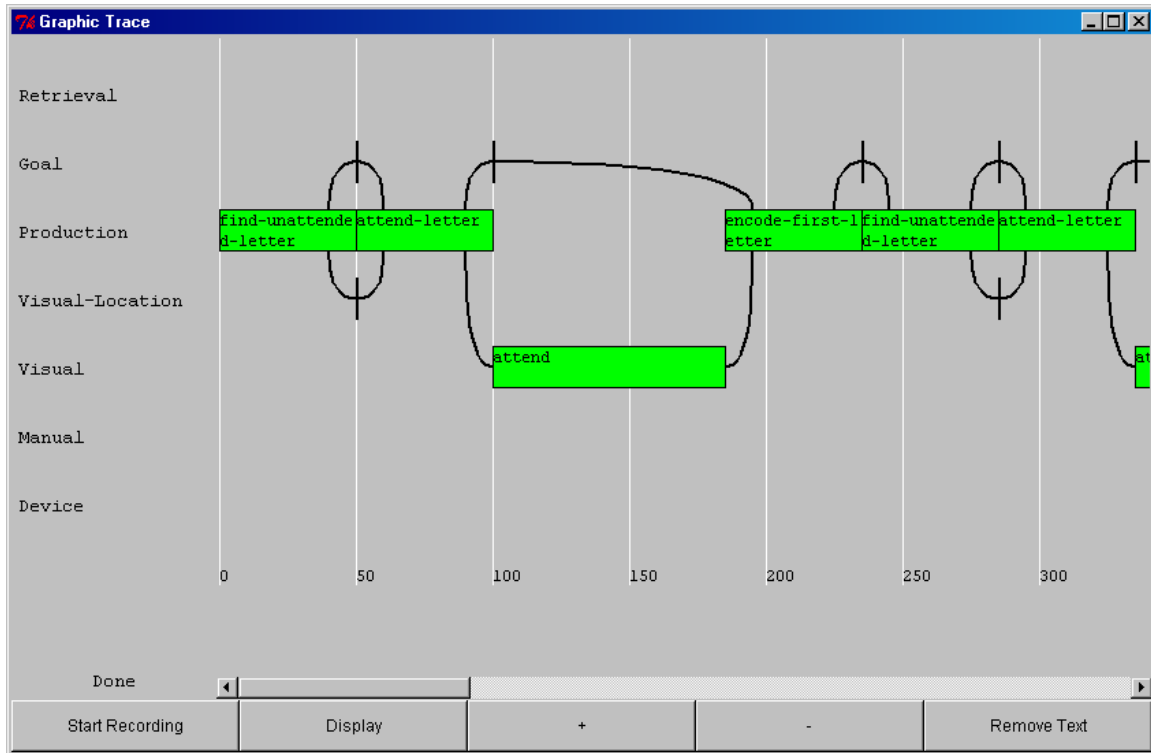
Pressing the “Graphic Trace” button opens up a window that allows you to see a graphic trace of the dependencies of the modules for a model run (something like a PERT chart). It is still in its early stages of development, and is fairly slow to draw and by default doesn’t have any way to save the results (though it is possible to add a new button to the

control panel that will save the graphic trace window as a postscript file – see the information in the technical details below).

This is what the window looks like initially:



and this is how it will look when there is a trace of a model displayed :



How to read the trace

Each of the main modules of ACT-R 5 is represented on a row of the diagram as well as a row for the external device. Time advances along the horizontal axis and a scale marked at 50ms increments is displayed. A box on a row (other than the Production's row) indicates that that module has had a request made to it and it is busy during the time spanned by the box (which may be zero seconds for some of the requests, particularly Goal and Visual-location requests). The production row shows a box for each of the productions that fire, which is similar to the other rows, but the boxes are not initiated by a request (the procedural system essentially runs continuously and fires productions as they match with the restriction that only one can fire at a time). The manual row is a little different as well. That row is split in half with the top half representing the preparation stage and the bottom half representing the execution stage of motor processing. This is done because it is possible that the stages are operating on different actions simultaneously and that needs to be discernable in the display.

The text displayed in the boxes represents the request that was made, with two exceptions. The retrieval row shows the result of the retrieval, which is either the name of the chunk retrieved, the failure indicator, or an aborted retrieval indicator (a retrieval request is aborted if a new one is initiated before the previous one completes). The production row shows the name of the production that fires in the box.

The lines represent the conditions for the productions and the actions that they request. A line from the right end of a production to one of the modules indicates that that production made a request to that module's buffer. A line from a module row into the left of a production box indicates that that production tested that particular module's buffer. There are also lines that project to the external row which indicate a module initiated an action that affects the external device. One set of lines that are missing at this point and should probably be added are those from the external device to the modules to represent things like buffer stuffing, but those are not really available in the normal trace at this point so they have not been added to the graphic trace.

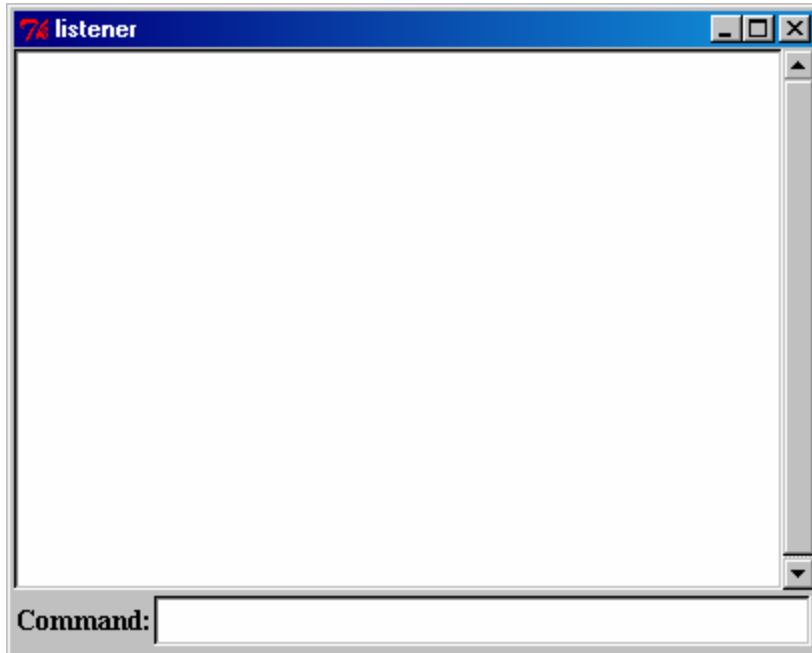
How to use the Graphic Trace Tool

Open the Graphic Trace window. Then press the "Start Recording" button. That will remove any trace that may already be in the display and start recording all of the events generated by ACT-R. Now you should run your model for as long as you would like it to be traced. When that is done you should press the "Display" button to have the trace of the recorded events drawn. The drawing is not very fast at this point, and you should wait for it to complete before manipulating the view (there is an indicator over the "Start Recording" button to indicate whether it is busy or done). If you then continue to run the model the new events will also be recorded, and pressing the "Display" button will result in the entire trace being drawn starting at the time when you last pressed "Start Recording".

Once the display is completed you can use the "+" and "-" buttons to zoom in or out respectively. Pressing the "Remove Text" button will remove all of the text from the display which can be useful if it is overly cluttered, but the only way to get it back once it has been removed is to redisplay the entire trace.

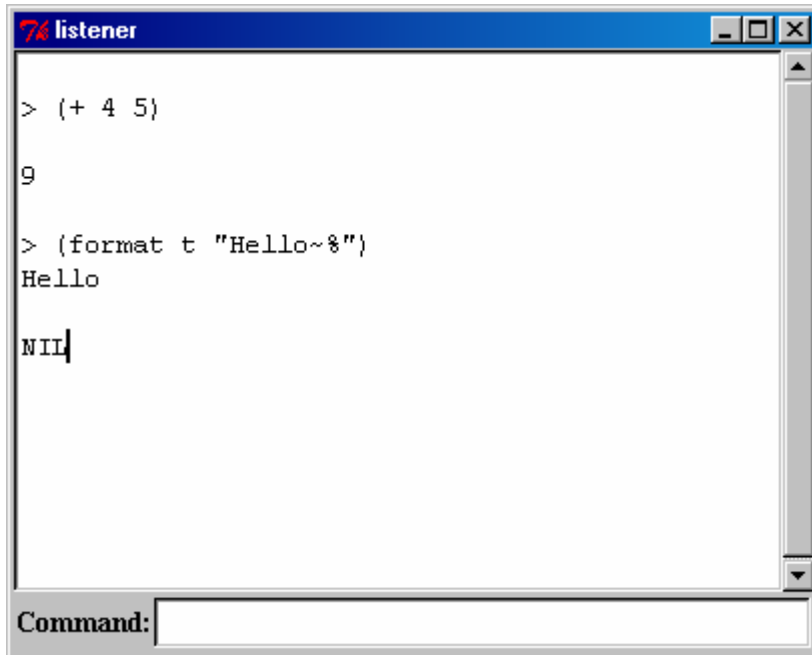
Standalone version

The standalone version works very much like the “full” version, but there are some differences. The biggest difference is the inclusion of a Listener. The standalone environment comes with a built in Lisp image which you interact with through the Listener window. This window looks like this:



It operates similar to a typical listener window in a Lisp, but is a bit more restrictive. It is where text printed to **standard-output** will be displayed (that includes the model’s trace and !output! commands from the model), and provides the input for **standard-input** (though I haven’t tested that thoroughly so you may get some unexpected results and let me know if that happens).

All of the commands you want to execute (or text to send to **standard-input**) must be entered in the Command area at the bottom of the window and cannot span multiple lines i.e. as soon as you hit Enter the command is sent to be executed whether it is a valid form or not. The command you enter will be copied to the upper text window portion of the Listener and it will be followed by any text display from the execution and then the return value of the call. Here is an image of the listener window after a couple of simple commands have been executed:

A screenshot of a window titled "listener" with a blue title bar. The window contains a text area with the following text:

```
> (+ 4 5)
9
> (format t "Hello~%")
Hello
NIL
```

At the bottom of the window, there is a "Command:" label followed by an empty text input field.

There is one other restriction (that I know of) on commands that can be entered. Commands with backslash characters “\” don’t work right yet (it’s an escape character in Tcl as well as Lisp and I should be able to fix it for a future release). Thus, you can’t do something like this (format t “~c” #\a), but in most cases you can work around that by using a double backslash like this (format t “~c” #\\a) until I fix that.

Another difference with the standalone version is that you don’t connect/disconnect it from a Lisp because it’s always connected, and you don’t quit the application by closing the control panel. Instead, to quit the standalone environment you close the listener window. It will prompt you if that’s really what you want to do and if you respond positively it quits the application.

As for the Lisp that’s “underneath” the standalone version, in Windows it’s a distributable executable image built from ACL 6.2 and in MacOS X it’s basically a complete version of OpenMCL. As such, the Windows version does not contain a compiler, a debugger, or any of the fancy ACL features but should be sufficient for someone learning ACT-R. The reason that I can distribute OpenMCL like that is because it is distributed under the LGPL (at least that’s my understanding of the situation after contacting the author – if you feel otherwise please let me know).

Known Bugs/Issues

- The following problem has been “fixed”, but if you are experiencing any problems with it (other TCP/IP sockets are misbehaving now) please let me! ~~When it is connected to MCL there will be a significant slow down in performance even in the absence of there~~

~~being any active inspectors. Only MCL suffers in this way (ACL, LispWorks, and OpenMCL have a virtually unnoticeable slowdown as long as the environment is idle). The reason is because the Opentransport stream is locked if I let it block on a read and that prevents me from being able to write to it, so instead I have to constantly poll the connection. There may be a way to “fix” that, but for now when using MCL one should always stop the environment when it’s not being used.~~

- If you have any open windows on the environment side it will cause a significant slow down of any model running because there’s likely to be some network traffic on the hook functions. I don’t consider that much of a problem though because if you’re inspecting something chances are you don’t want it running at full speed anyway, but if people have a problem with that I can work to “improve” the way things interact.

- In ACL 5.0 and 5.0.1 you can't use the open model button (it will cause a Lisp error and fail to open the file or worse it'll open it ok, but when you try to save it will mangle the model file). The problem is that in ACL 5 different functions return/require different file positions because some count a CR/LF pair as only one character (incorrectly). That’s something that could be worked around for the old environment, but isn’t “hacked” into the new environment.

- In OpenMCL if you turn on the compile file option you will have problems opening model files. The default for the Mac version is with that option being off, and as long as one leaves it there things are fine.

- In ACL don't turn off the use environment window option or it will cause problems for the model because it will fall back to using an ACL window (assuming you’re in an IDE’ed version of ACL) but if you’re using the stepper the experiment window won’t have the focus when the model “interacts” with it and the responses will not get handled properly. [In fact, right now one can’t turn it off in any version.]

- If the environment connection is terminated “incorrectly” (most likely to occur if one quits or kills the environment) the Lisp may hang, and with MCL I’ve found that it may hang the machine as well. If it doesn’t hang, you may end up in a state where the Lisp will complain if you try to call **start-environment** saying that there is already a connection. If that happens, to connect to the environment without restarting the Lisp you need to call **close-all-connections** first to clean things up.

- The stepper is not yet designed for use on more than one Environment at a time in a multi-Environment setup. Only one of the machines should have a stepper open at any time, otherwise they will end up out of sync.

- The items in the “Running...” section of the Control Panel are designed so that you can’t use one while another is still “pending”. However, if an error occurs during the execution of one it may leave the system thinking that it’s still “pending”. If you encounter such a situation (an error occurs when you use one of these buttons, and

subsequent attempts to use that or other buttons seems to have no effect) then in Lisp, you should check the state of the global variable `*running-actr*`. If it is t, then if you clear that by setting it to nil (`setf *running-actr* nil`) you should be able to use the buttons again. [I think I catch all of these with `unwind-protect` and `ignore-errors`, but there were problems in the past, so just to be safe I'm leaving this note in.]

- There have be reports of problems with the stepper when using the “old style” `!output!` command - using the format directives in the string followed by variables like this:

```
!output! (“This is a chunk ~S” =var)
```

I haven't replicated the problem yet, so I don't have a fix for it directly, but if those are converted to the new style `!output!` commands there isn't a problem i.e.

```
!output! (This is a chunk =var)
```

- When using Tcl/Tk windows for the AGI it's sometimes the case that they are not “brought to the front” correctly when running with a Lisp (as opposed to the standalone version) because the Lisp application's windows obscure them. The easiest workaround is to not position windows where the AGI window will be (or create the AGI window in open space) so that they are not obscured. This is a tricky issue to fix cross platform, and depending on your OS and Lisp you may or may not experience such problems.

Some Design notes

The biggest change is the use of Tcl/Tk [Tcl is a cross platform scripting language and Tk is a GUI toolkit that works with Tcl] to provide the GUI instead of using the GUI elements of particular Lisps. This introduces some complication because now there are essentially two applications running – the Lisp that runs ACT-R and the Tcl/Tk running the Environment, but I don't think that that introduces too much extra difficulty and I think the benefits far outweigh that. The first benefit is that the environment is now almost completely independent of the Lisp software being used and thus there is only one Environment – it's the same source code on all platforms. Everybody is using the same one no matter what Lisp or OS they are using, and it's not tied directly to specific versions of particular Lisp implementations so it's going to be much easier to keep up to date. This also means that we can support a wider range of Lisp applications more easily (see the tech info at the end for the details of adding a new Lisp) and right now it is available to a wider audience than the old Environments ever were (see above for systems and Lisps currently supported). This also means that those with some knowledge of Tcl/Tk can customize the environment to better suit their needs (by either modifying the current tools or adding whole new ones), and it has been written so that even those without knowledge of Tcl/Tk can at least remove unnecessary elements from the control panel to make it easier to navigate.

I see the new Environment being used slightly differently than the previous ones as well. I see it as a separate tool for debugging and teaching, not a monolithic system for running ACT-R like the old one was perceived to be (though the standalone versions will still exhibit that character). When you need it, you connect it to the Lisp that's running ACT-R and when you no longer need it you can disconnect it and go back to what you were doing. ACT-R is very much a Lisp based tool, and the Environment is not an attempt to move the focus away from Lisp or to hide the Lisp with a GUI – if a user isn't comfortable with a Lisp prompt for interacting with ACT-R the Environment doesn't really do much to alleviate that. As far as the provided tools go, the “open model” options are there as a continuation of the old system, but I feel that they should only be used by students early on. After that, people should move to using a better editor (the editor provided by their Lisp application, Emacs, or just about anything else is going to be a better editing tool than the Environment's windows) and operate more independently of the environment. After unit 1 it's not likely that the run button does much good either, and perhaps there should be an option setting that switches between novice and “regular” user such that for a regular user the open and running options are automatically removed (as opposed to the user “removing” them by hand).

That's all I'll go into here, but there are lots of other things that can be discussed about the new environment, so feel free to contact me. [If you were at the 2001 ACT-R post-graduate summer school you probably remember that when it comes to the Environment I can talk forever!]

Technical Details

Configuring the Control Panel

The control panel is built from the Tcl files in the GUI/dialogs directory. All of the files with a .tcl extension are sourced in sorted order. That's why the names have the numbers on the front. The names (hopefully) indicate what functionality is provided in each file. If you want to rearrange the order of the items all you need to do is change the file names so that they are ordered as you want. If you want to remove an item all you need to do is either delete or rename the corresponding file. Most of the items are independent so you can remove one without affecting the operation of the rest, but there are some exceptions (the open-button has procedures that are used by other buttons) which I haven't documented yet so you may not want to delete a file until you try the environment without it first.

Adding new Functionality to the Environment

Because all of the .tcl files from that directory are sourced it's possible to add your own functionality to the environment simply by writing the Tcl scripts and placing them there. I don't have much documentation yet on how one can use the internals of the environment for contacting Lisp from Tcl or how you'd put a button on the control panel,

but the current tools should provide reasonable examples and in particular the declarative viewer was heavily documented for use as an example.

There are actually some extra buttons already included which are not placed onto the control panel. They have a .tcx extension in the distribution but if they are renamed with .tcl at the end they will be used. The two extra buttons are a “Save As” button (called 15-ctrl-panel-save-as-button.tcx) which will allow one to save a model file under a different name and a button called “Save Trace” (the 96-ctrl-panel-print-gt.tcx file) which will allow you to save the Graphic trace window to a postscript file.

Adding support for a new Lisp

In theory, the only thing that needs to be done is to edit the uni-files.lisp file and add a version of each of the functions in that file that works in the new Lisp and change the loader.lisp file if necessary to get the pathing and compiling set right for the new Lisp. I say in theory because there may be some minor issues that would call for the need to change some of the other Lisp files, but the design is such that that *shouldn't* happen.

The functions that need to be added are ones that deal with opening, closing, and managing a stream over a socket, ones for starting and stopping a process in a different thread, a function to handle errors that occur, one to allow system events to process, and one to coerce a cons to a function if necessary. They are all documented in that file, and it shouldn't be too difficult for an experienced Lisp programmer to copy/modify the existing versions (or maybe even improve upon them) for use with a new Lisp. The estimate I've been working with is that I expect it'll take me about a day's work to add and debug the support for a new Lisp - assuming it supports everything necessary and I've got the appropriate docs. Of course everybody knows how accurate software design estimates are, but that's my claim at this point. [Note, it took me a little less than 2 days to add LispWorks, so that estimate of about a day isn't too far off.]

I think that should be enough to get someone started on supporting a new Lisp, and feel free to contact me with questions if you want to try adding one.