QuickLink Tutorial

by Taylor Lafrinere

1 Introduction

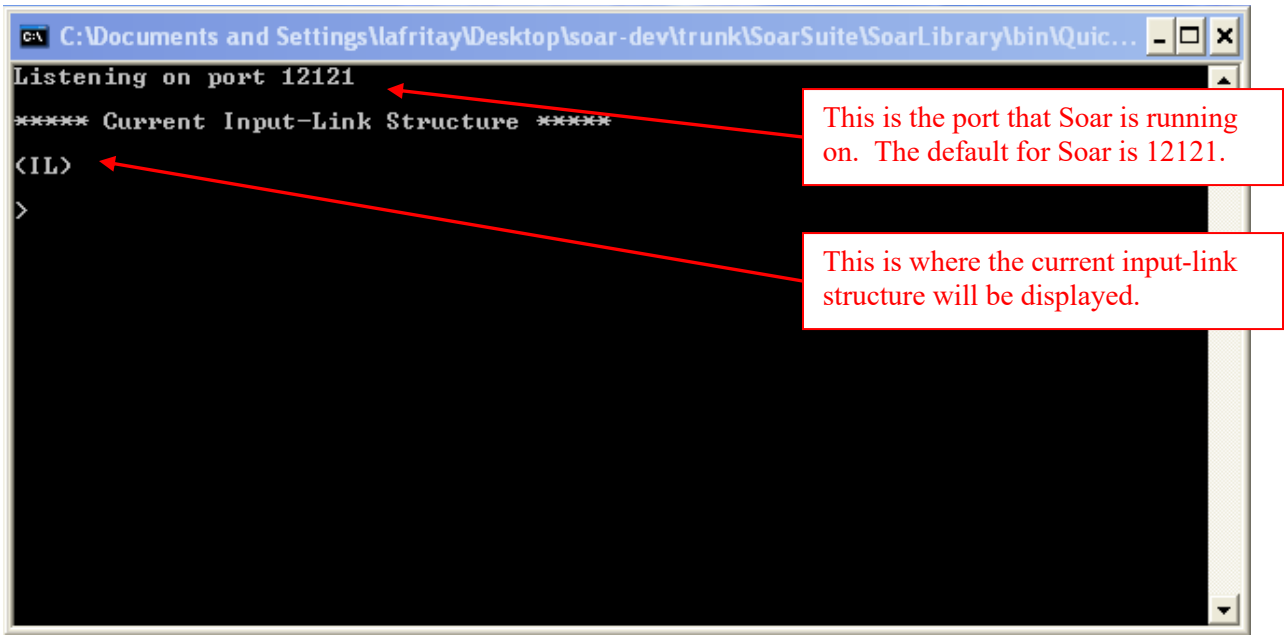
The purpose of this tool is to give complete control of the input-link to the Soar programmer. QuickLink can be thought of as a way to "fake" an external application, such as a game or simulation, in order to test specific circumstances that are rare and/or hard to reproduce. QuickLink currently can only be operated through the use of a command line interface.

2 Getting Started

2.1 Launching

The QuickLink executable should be located in <soar-dir>/SoarLibrary/bin/ and is appropriately named "QuickLink." Once you find the icon, double-click it to launch the program. The QuickLink (QL) command-line window will pop-up. On launch, QL will internally create its own instance of Soar along with an agent named "QuickLink."

Here are some of the basics about the QL command-line:

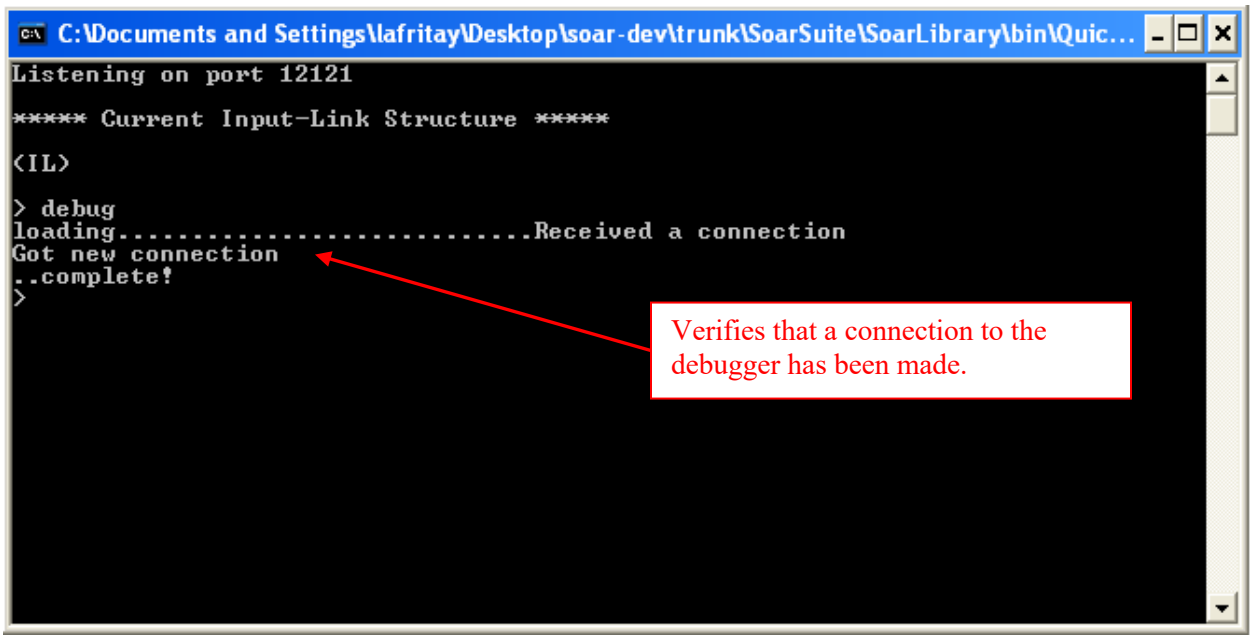


QL has a built-in command that allows you to spawn the SoarJavaDebugger and have it automatically attach to the QuickLink agent. To do this type:

```
debug
```

Spawns the SoarJavaDebugger and establishes a remote connection to QL.

The QL command-line should indicate that a new connection has been made. If for some reason the "loading…" text doesn't stop printing periods, there is most likely something wrong with the path information on you computer. First, make sure there are no other instances of the SoarJavaDebugger open on you system. Next, verify that both the QuickLink executable and SoarJavaDebugger jar file are both located in <soar-dir>SoarSuite/SoarLibrary/bin. If things still don't work and you are on Linux, make sure that your LD_LIBRARY_PATH environment variable is set correctly. **Note:** QL does not need to be attached to the debugger in order to run, it just makes it easier to see what is going on.



QL is now ready to start simulating an external environment. At any time we could start adding WME's to the input-link and running Soar, but that wouldn't be any fun because there are no productions loaded. So in order to make this interesting, let's load a TankSoar agent's productions. This can be done either via the QL command-line or through the SoarJavaDebugger (**Note:** All Soar command-line commands can be run through the QL command-line interface. Any command that QL doesn't recognize will be sent to the Soar command-line. This will be useful later when we talk about scripting in QL).

In the instance of the SoarJavaDebugger that is connected to QL, click File | Load Source File …. A window will pop up to the location that you last loaded Soar productions from. Navigate to the SoarSuite/Environments/JavaTankSoar/agents/tutorial folder and double-click on mapping-bot.soar (mapping-bot is used for this tutorial as opposed to the simple-bot because it is able to make decisions on less detailed input-link structures, which will make this tutorial less tedious). The productions you have just loaded are from the game TankSoar (which you should know about if you have read the Soar

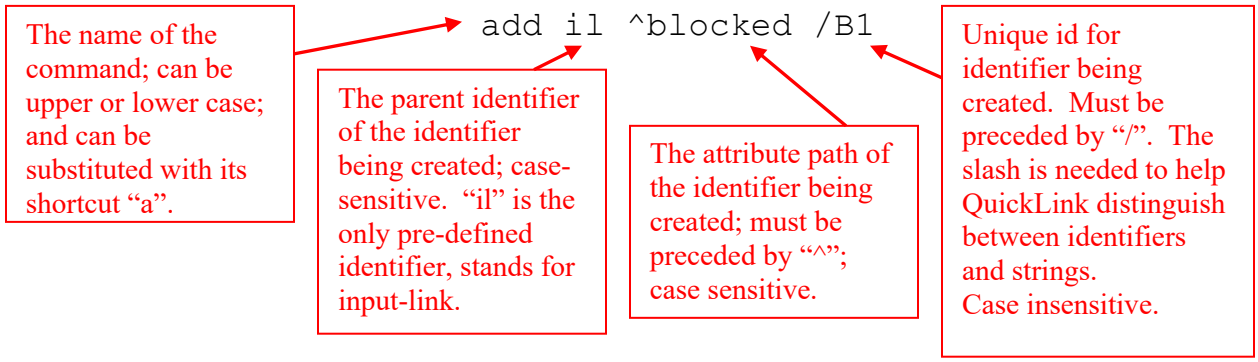
Tutorial). We will use these productions to demonstrate how QuickLink can be used imitate an external application such as TankSoar.

3 Creating Elements

All commands used in QuickLink follow the same basic structure and have been constructed to model Soar conventions. All commands have a full name and a shortcut name such as "add" and "a" which can be used interchangeably.

3.1 Adding Identifiers

All commands have a command name which is always one word (containing no white-space but is sometimes multiple words combined together) which is followed by the appropriate number of arguments. For example, this is the command used to add an identifier to the input-link



This command will add an identifier to the input-link (il) with an attribute path (blocked) that has its own case-insensitive identifier name (B1). To do this, type the add command shown above into the command line window and press enter. You should see the following changes:

```
C:\Documents and Settings\lafritay\Desktop\soar-dev\trunk\SoarSuite\SoarLibrary\bin\Quic...
Listening on port 12121

***** Current Input-Link Structure *****

(IL)

> debug
loading.............................Received a connection
Got new connection
..complete!
> add il ^blocked /B1

***** Current Input-Link Structure *****

(IL ^blocked B1)
  (B1)

> _
```

Notice that the '/' sign has been removed before B1, it is only used to indicate that the element is an identifier at the time of creation. In other words, the unique id for blocked is 'B1' not '/B1'.

Notice that as soon as enter was pressed the input-link structure was updated and re-printed. (But also note that the agent will not see these changes until its next input phase). But for what reason did you add an identifier with an attribute path of blocked to the input-link? The answer is that since we are trying to imitate the game TankSoar, we have to put the same commands on the input-link that TankSoar would. "blocked" is an identifier that has four directions (forward, backward, left and right) that is used to tell the tank which spaces around it are blocked and unblocked.

The entire input and output-link structures have been included on the next page for reference. This is a map of all of the possible structures that could exist on either of these links, note that they do not all have to exist for the program to function. I would suggest printing the next page so that you can reference it as the tutorial progresses.

(Note that if you make a mistake in adding things to the input-link, see Section 5.5 on deleting wmes).

Map of Input-link and Output-link

```
^io
   ^input-link
      ^blocked
         ^backward yes/no
         ^forward yes/no
         ^left yes/no
         ^right yes/no
      ^incoming
         ^backward yes/no
         ^forward yes/no
         ^left yes/no
         ^right yes/no
      ^radar
         ^energy
            ^distance 0-13
            ^position left/center/right
         ^health
            ^distance 0-13
            ^position left/center/right
         ^missiles
            ^distance 0-13
            ^position left/center/right
         ^obstacle
            ^distance 0-13
            ^position left/center/right
         ^open
            ^distance 0-13
            ^position left/center/right
         ^tank
            ^distance 0-13
            ^position left/center/right
```

```
^io
   ^input-link
      ^color red/blue/purple/…
      ^rwaves
         ^backward yes/no
         ^forward yes/no
         ^left yes/no
         ^right yes/no
      ^smell
         ^color none/red/blue/purple/…
         ^distance none/0-28
      ^sound silent/left/right/
             forward/backward
      ^clock 1-N
      ^direction north/east/south/west
      ^energy 0-1000
      ^energyrecharger no/yes
      ^health 0-1000
      ^healthrecharger no/yes
      ^missiles 0-N
      ^my-color blue/red/purple/…
      ^radar-distances 1-14
      ^radar-setting 1-14
      ^radar-status on/off
      ^random 0.0-1.0
       ^resurrected no/yes.
       ^shield-status on/off
       ^x 1-14
       ^y 1-14
```

```
^io
   ^output-link
      ^move.direction left/right/forward/backward
      ^rotate.direction left/right
      ^fire.weapon missile
      ^radar.switch on/off
      ^radar-power.setting 1-14
      ^shields.switch on/off
```

3.2 Adding Elements With Values

The syntax for adding an element with a value is very similar to adding an identifier. The only difference between the two is that the last argument is not preceded by a '/' symbol. Look at this example:

```
add il ^blocked /B1
```

Creates an identifier on the input-link with the unique id B1

```
add il ^blocked B1
```

Creates a string on the input-link with a value "B1"

This is the most important lesson to learn about adding structures to the input-link: To create identifiers, always include a '/' sign before the last argument and to create value-based structures (strings, integers and floats) just simply type the value. (Note: the names used as unique id's given to identifiers are completely arbitrary, although it is a Soar convention to use a capital letter followed by a number, for example, 'B1'. These unique id's will probably not match the actual identifiers that Soar will create in working memory, but merely serve as QuickLink's way to refer to them).

If you look at your map of the input and output-link structures from TankSoar, you will see that under the identifier "blocked" there are four elements (backward, forward, left and right). Each of these elements has the possible values of either "yes" or "no." Let's add these elements to the input-link. The syntax for adding the element with the attribute path "backward" would be:

```
a B1 ^backward no
```

This time the shortcut for add which is "a" is used

Since we want backward to be located under blocked, we set the parent identifier as B1

The attribute path, just as before

The value we want this element to have. Represents that there is an open space behind the tank. Note that string values like this are case-sensitive.

Type the above command into QuickLink and press enter, you should see "^backward no" added to B1. Add the following commands to represent the rest of the blocked structure:

```
a b1 ^forward yes

a B1 ^left no

a b1 ^right no
```

Notice here that we used "b1" instead of "B1". Remember that identifier names are case-insensitive, so either can be used.

After entering in these commands, your screen should look as follows:

```
C:\Documents and Settings\lafritay\Desktop\soar-dev\trunk\SoarSuite\SoarLibrary\bin\Quic...
(IL ^blocked B1)
  (B1 ^backward no)

> a b1 ^forward yes

***** Current Input-Link Structure *****

(IL ^blocked B1)
  (B1 ^backward no ^forward yes)

> a B1 ^left no

***** Current Input-Link Structure *****

(IL ^blocked B1)
  (B1 ^backward no ^forward yes ^left no)

> a b1 ^right no

***** Current Input-Link Structure *****

(IL ^blocked B1)
  (B1 ^backward no ^forward yes ^left no ^right no)

> _
```

All of the directions have been added to the identifier B1.

The current input-link structure now represents a tank that has a barrier of some sort in front of it, but has open spaces to both its sides and behind it.

4 Running Soar

With the structures we have added to the input-link we have supplied the mapping-bot productions loaded in the Soar Debugger with enough information to take an action. QuickLink has two different ways control can be handed over to Soar.

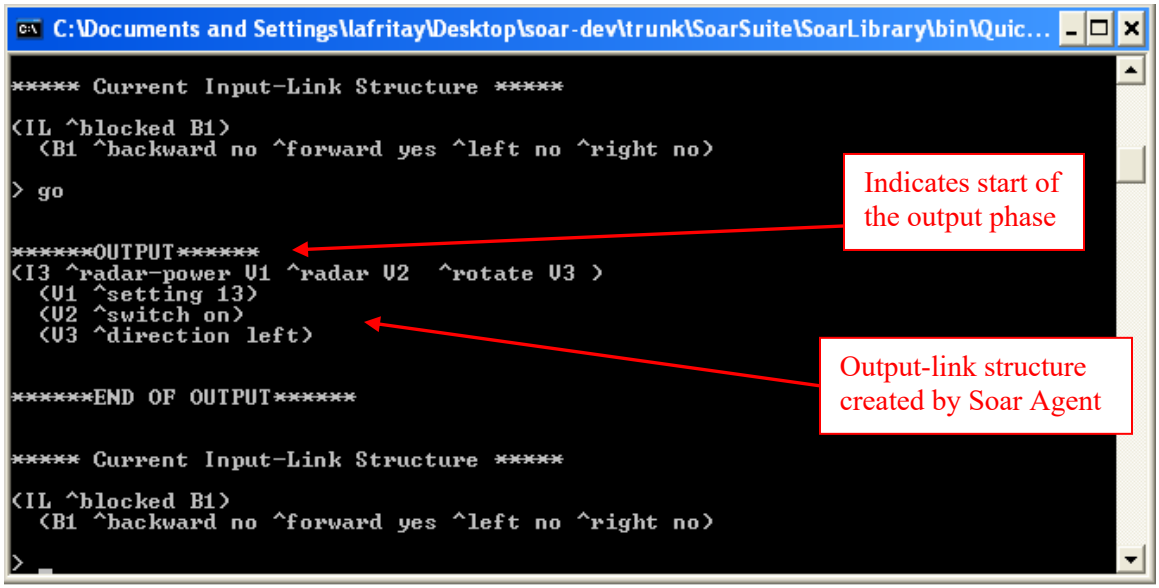
4.1 Running Until Output

The recommended way of running TankSoar agents once the input-link structure has been created is to tell the agent to run until it produces output. This can be done by typing `go` or `g` in the command line (Note: `go` is simply an alias for the Soar command `run -o` which could have also been used here). The debugger runs just as if it were hooked up to the actual TankSoar application. After the Soar Agent has processed the information on the input-link, it responds by placing commands on the output-link. These commands are then displayed in the command window in the same format as the input-link structure.

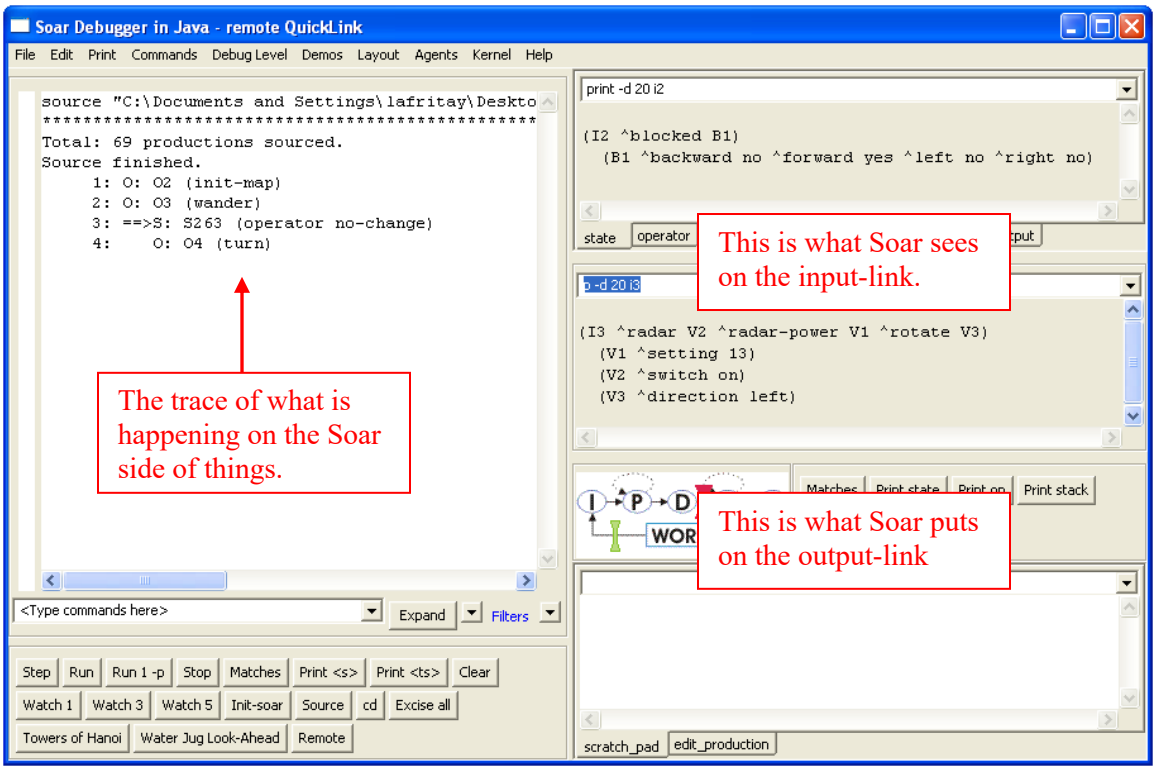
4.2 Run a Cycle

You can also use the standard Soar `run` command to run a fixed number of cycles (or to run in other ways). To run a fixed number of cycles, the syntax would be `run <num>` where <num> is an integer that indicates the number of cycles to be run.

Type `go` into the command line and press enter. Your display should look as follows:



You should also notice that the Main Trace window in the Debugger has changed to reflect what has just happened.



If you look at the map of the input and output-link structures you will see that this output looks similar to the map of the output. This output structure is giving the following commands:

1. Set the radar-power to 13.
2. Turn the radar on.
3. Rotate left.

Now the beauty of QuickLink is that you do no have to obey these commands if you do not want to. You can make any changes to the input-link that you see fit. The downside though is that you have to make these changes yourself, they are not done automatically. For the purpose of this tutorial, let's do our best to follow the output commands.

Note: As you continue to use QuickLink you may notice that the output it displays is different from the output displayed in the debugger. This is because QuickLink only displays the things that have *changed* on the output-link since the last time it was displayed. This prevents the confusion of trying to determine if an output-link command is new or is leftover from earlier.

5 Making Changes

Now let's change the input-link to reflect what has been put on the output-link. The first command was to set radar-power to 13. If you look on your TankSoar map of the input-link in right-hand column towards the bottom you will see a structure off of the input-link titled radar-setting which is followed by an int from 1-14. Directly below that is an element called radar-status which corresponds to the second output-link command. Add the following commands one at a time so that we can make these elements active on the input-link:

```
a il ^radar-setting 13

a il ^radar-status on
```

You should see these two structures appear on the input-link. Now is a great time to show how specific instances can be tested. If you look once again at the input-link map, you will see that the entire bottom of the left side is dedicated to perceiving other objects using the radar identifier (which fits in with this example because we just turned radar on). Let's give the tank some input on the objects around it to see how it reacts.

First, we have to create the radar identifier which is found on the input-link:

```
a il ^radar /R1
```

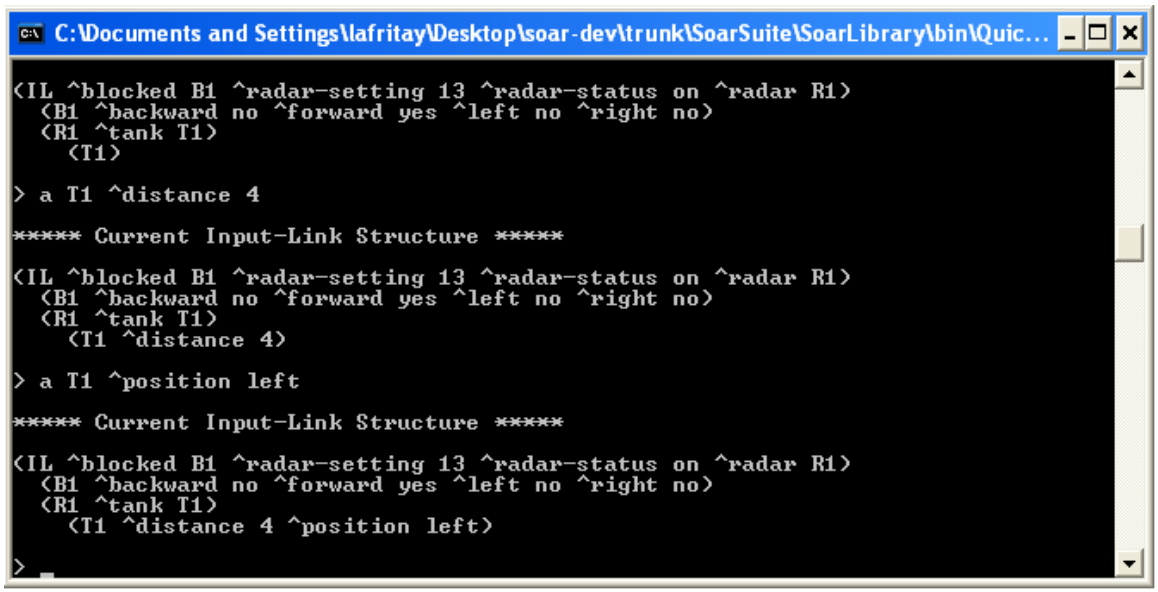
Next, let's tell the tank that there is an enemy tank four spaces in front of it and off to the left. This can be done using the following commands:

```
a R1 ^tank /T1

a T1 ^distance 4
a T1 ^position left
```

Creates a tank identifier off of the radar identifier

Puts information about location of the tank on tank identifier

Be sure to notice that we are following the structure of the input-link map. You screen should now look like this:



5.1 Retrieving Last Output

Now we have taken care of two of the proposed commands from the output-link, but remember that there were three. What I don't remember is what that command was. Now to make things worse I can't see the output-link print anymore. What should I do? Luckily, QuickLink saves what was printed to the output-link so that you can view it again at any time. This is done by typing "output" (which is not case-sensitive). Do this now to see what command we were forgetting. When the output prints again you will see that we have been ordered to rotate left.

5.2 Altering Elements on Input-Link

The tank has requested that we rotate it left. In order to do this we need to alter the forward element on the blocked identifier to say no and the right element on the blocked identifier to say yes, this is equal to a rotate left. Changing a pre-existing element has its own syntax in QuickLink which is as follows:

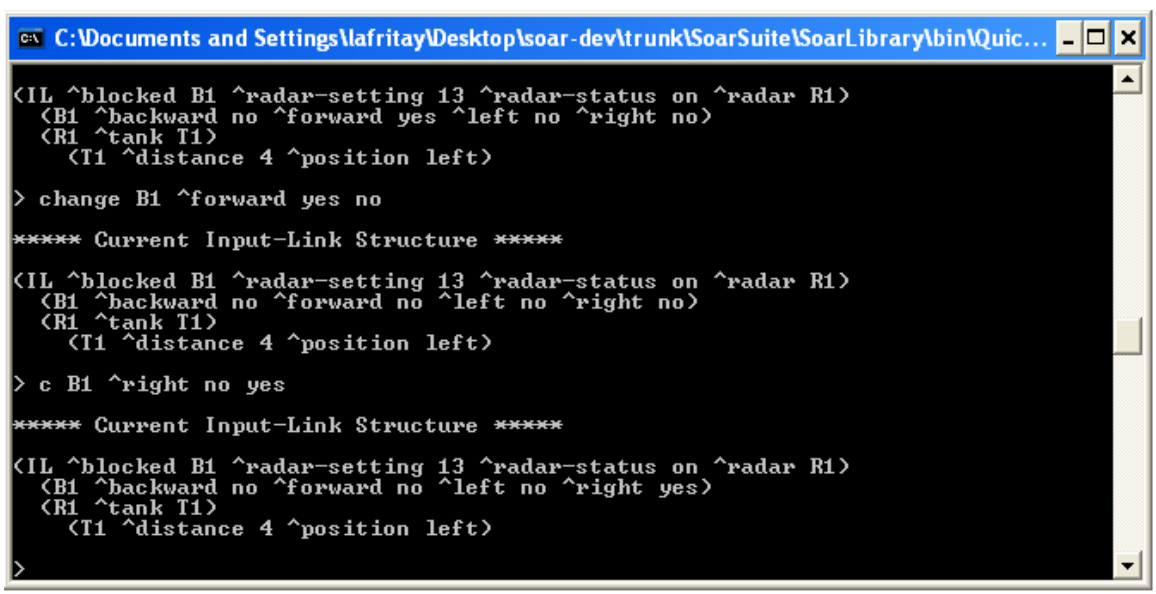
```
change <parent-id> ^<path> <old value> <new value>
```

- You can either use change or 'c'.
- Variable for parent id. Angle brackets represent variables.
- Variable for path. Angle brackets represent variables.
- Variables for old value and new value. Angle brackets represent variables.

Enter the following commands to change the forward and right elements which are located on blocked:

```
change B1 ^forward yes no

c b1 ^right no yes
```

Your screen should look as follows:



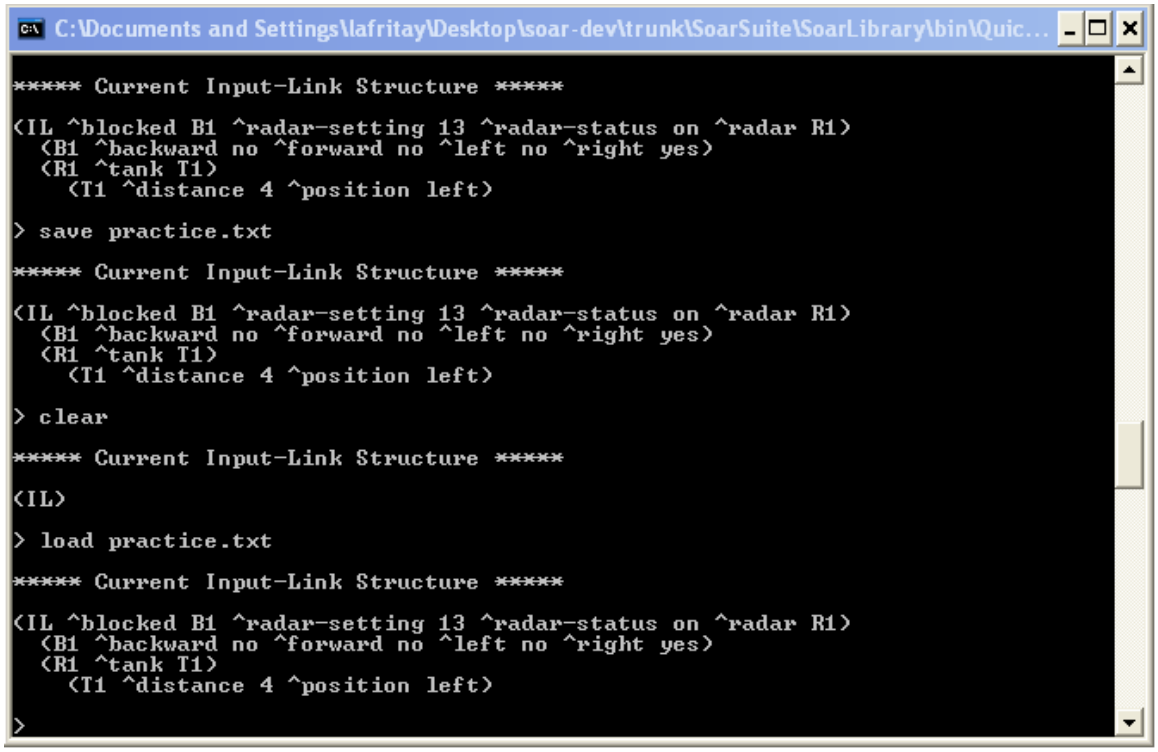
5.3 Saving and Loading A Structure

Adding and modifying all of those elements was a lot of work. QuickLink allows you to save and load the current input-link structure so that you are not risking losing all of your work and so that you can load them later to save time. Let's practice this.

Type `save practice.txt` and press enter. This saves your current input-link structure to the file practice.txt in the current working directory, which is probably <soar-dir>\SoarSuite\SoarLibrary\bin.

Now type `clear` to erase everything currently on the input-link.

Now type `load practice.txt` to reload your input-link structure. (**Note:** Doing a save, followed by a clear, followed by a load will actually blink the input-link structure. This means that on the Soar side of things, these are actually all new elements, which will cause rules to retract and fire again.) Your structure should be just as you left it and the trace should look like this:



5.4 More Running

Now let's see what happens. Type `go` just as before and press enter. Your screen should look like this:

```
C:\Documents and Settings\lafritay\Desktop\soar-dev\trunk\SoarSuite\SoarLibrary\bin\Quic...

(IL ^blocked B1 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right yes)
  (R1 ^tank T1)
    (T1 ^distance 4 ^position left)

> go


******OUTPUT******
(I3 ^move U4)
  (U4 ^direction left)


******END OF OUTPUT******


***** Current Input-Link Structure *****

(IL ^blocked B1 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right yes)
  (R1 ^tank T1)
    (T1 ^distance 4 ^position left)

>
```

Command indicating that the tank wants to move to the left.

The Soar Agent has instructed the tank to move left via the output-link. It is now time to make some more changes. Think about why this has happened. We have simulated the TankSoar environment to make our tank think that there is an enemy tank 4 spaces ahead of him and one space to the left. The tank is ordering us to move left so that it will be able to shoot the enemy tank. In order to move the tank left here is what needs to be changed:

1. The tank is no longer blocked on the right.
2. The enemy tank now has a position of center.
3. We should give the tank missiles so it can shoot the other tank.

Let's make these changes to see how the tank responds.

5.5 Deleting

While it makes sense to use the change command to alter the values needed to move the tank left, we are going to delete them and make new ones so that you can learn how to delete. The delete command is very similar to add command:

Warning: Do not enter these next two delete commands; they are only shown for comparison to the add commands.

Adds identifier B1 with path blocked to the input-link.

```
add il ^blocked /B1
```

```
delete il ^blocked /B1
```

Deletes identifier B1 with path blocked to the input-link.

The only difference between add and delete is the first word

Adds value yes with path right to the identifier B1.

```
add B1 ^right yes
```

```
delete B1 ^right yes
```

Deletes value yes with path right to the identifier B1.

Let's try this, first delete ^right yes off of blocked and ^position left off of tank with the following commands:

"d" is the shortcut for "delete".

```
delete B1 ^right yes
```

```
d T1 ^position left
```

Now add in new elements with the following commands:

Adds ^right no to blocked identifier.

```
a B1 ^right no
```

```
a t1 ^position center
```

Adds ^position center to blocked identifier.

Note: We could have used the change command to get the same effect here, but using fewer steps. I used the delete command to demonstrate how it worked.

If you look at your TankSoar input-link map, you will see that about halfway down the right column there is an attribute named missiles which is followed by an integer. Let's add this to our input-link structure:

```
a il ^missiles 15
```

Your input-link structure should now look like:

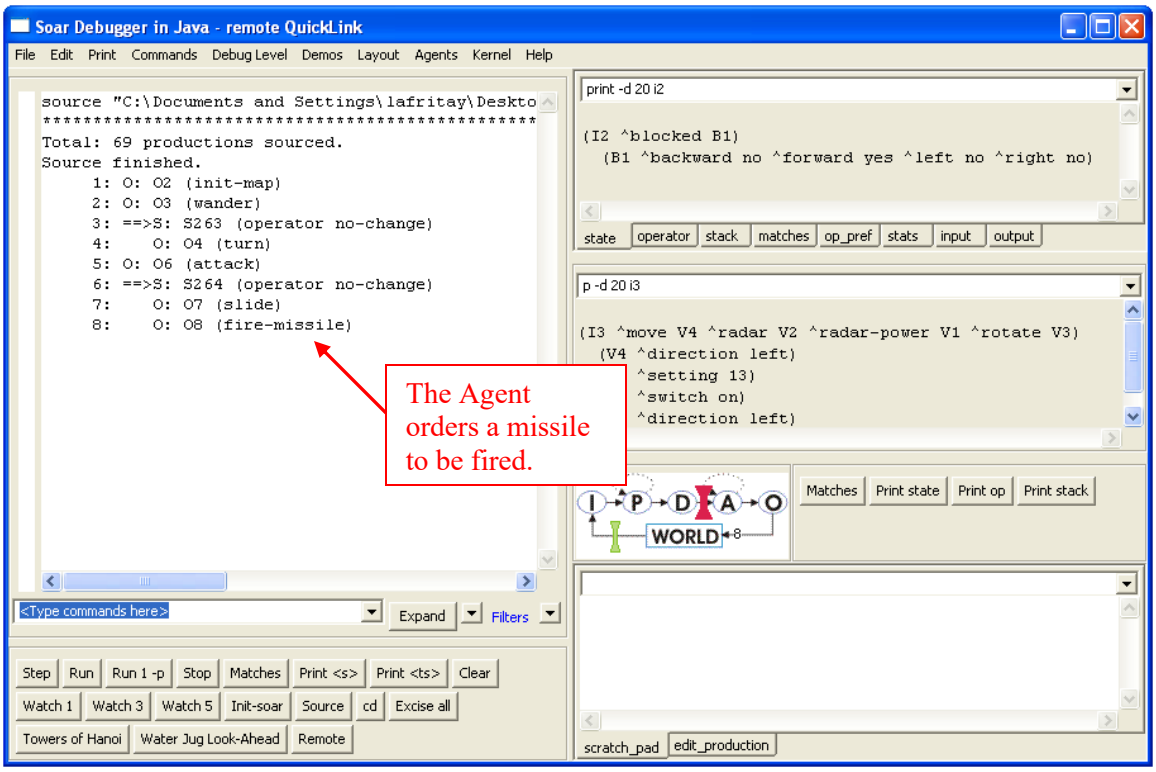
```
C:\Documents and Settings\lafritay\Desktop\soar-dev\trunk\SoarSuite\SoarLibrary\bin\Quic...

(IL ^blocked B1 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right no)
  (R1 ^tank T1)
    (T1 ^distance 4)

> a t1 ^position center

***** Current Input-Link Structure *****

(IL ^blocked B1 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right no)
  (R1 ^tank T1)
    (T1 ^distance 4 ^position center)

> a il ^missiles 15

***** Current Input-Link Structure *****

(IL ^blocked B1 ^missiles 15 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right no)
  (R1 ^tank T1)
    (T1 ^distance 4 ^position center)

>
```

Now type `go` to see how the Agent responds. You should see the following in the command window:

```
C:\Documents and Settings\lafritay\Desktop\soar-dev\trunk\SoarSuite\SoarLibrary\bin\Quic...

(IL ^blocked B1 ^missiles 15 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right no)
  (R1 ^tank T1)
    (T1 ^distance 4 ^position center)

> go

******OUTPUT******
(I3 ^fire V5)
  (V5 ^weapon missile)

******END OF OUTPUT******

***** Current Input-Link Structure *****

(IL ^blocked B1 ^missiles 15 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right no)
  (R1 ^tank T1)
    (T1 ^distance 4 ^position center)

>
```

The Agent orders a missile to be fired.

And if you look in the Debugger's Main Trace window:


Soar Debugger in Java - remote QuickLink
File Edit Print Commands Debug Level Demos Layout Agents Kernel Help
source "C:\Documents and Settings\lafritay\Deskto
****
Total: 69 productions sourced.
Source finished.
1: O: O2 (init-map)
2: O: O3 (wander)
3: ==>S: S263 (operator no-change)
4: O: O4 (turn)
5: O: O6 (attack)
6: ==>S: S264 (operator no-change)
7: O: O7 (slide)
8: O: O8 (fire-missile)
The Agent orders a missile to be fired.
<Type commands here>
Expand
Filters
Step Run Run 1 -p Stop Matches Print <s> Print <ts> Clear
Watch 1 Watch 3 Watch 5 Init-soar Source cd Excise all
Towers of Hanoi Water Jug Look-Ahead Remote
print -d 20 i2
(I2 ^blocked B1)
(B1 ^backward no ^forward yes ^left no ^right no)
state operator stack matches op_pref stats input output
p -d 20 i3
(I3 ^move V4 ^radar V2 ^radar-power V1 ^rotate V3)
(V4 ^direction left)
^setting 13)
^switch on)
^direction left)
I P D A O
WORLD
Matches Print state Print op Print stack
scratch_pad edit_production

6 Scripts

Now that you know the basic commands on how to add/delete/change structures on the input-link, it is time to learn about QL's most powerful aspect, that being the ability to run scripts. Since the main purpose of this program is to allow for quick and deliberate debugging, it would be nice if you didn't have to go through the steps of re-creating a whole set of input-link structures every time you wanted to debug code. This would be not only slow, but also very error prone. To combat this, QL creates scripts as you type in commands. For instance, all of the commands you have typed, from `debug` to the last time you typed `go` is sitting in QL's memory. All you have to do it save it to a file so that it can be run again.

6.1 Saving Scripts

The syntax for saving a script of commands is:

`script <filename.txt>` or `sc <filename.txt>`

Let's save the 4 steps we just did as a script. To do this, type

`script tanksoarQL.txt`

All scripts are saved in SoarLibrary/bin

6.2 Running Scripts

We just spent all this time creating and saving a script, so we might as well run it again. To do this, we first need to reset a few things. At any time, you can access the Debugger's command line directly through the QL command-line Let us use this to initialize the Soar Agent so it is ready to start running again. In the command window, type:

`init`

You should see a message notifying you that the Agent was reinitialized. Reinitializing an agent (as we just did) does not clear the input-link. To clear the input-link you must explicitly use the clear keyword as we did before. To clear the input link, type:

`clear`

Now it is time to load in the script that we just saved. To do this, type:

`load tanksoarQL.txt` or `l tanksoarQL.txt`

When you press enter your script will run and the TankSoar agent should repeat the behavior that it just went through.

6.3 Pause and Continue

Let's say you want to run a script that you have saved, but want it to run slower, or you want to make some change to the input link at an arbitrary point. In order to accomplish this, you can put various pause statements throughout your script. Open up the tanksoarQL.txt script we just made, it should look like this:

tanksoarQL.txt - Notepad

File Edit Format View Help

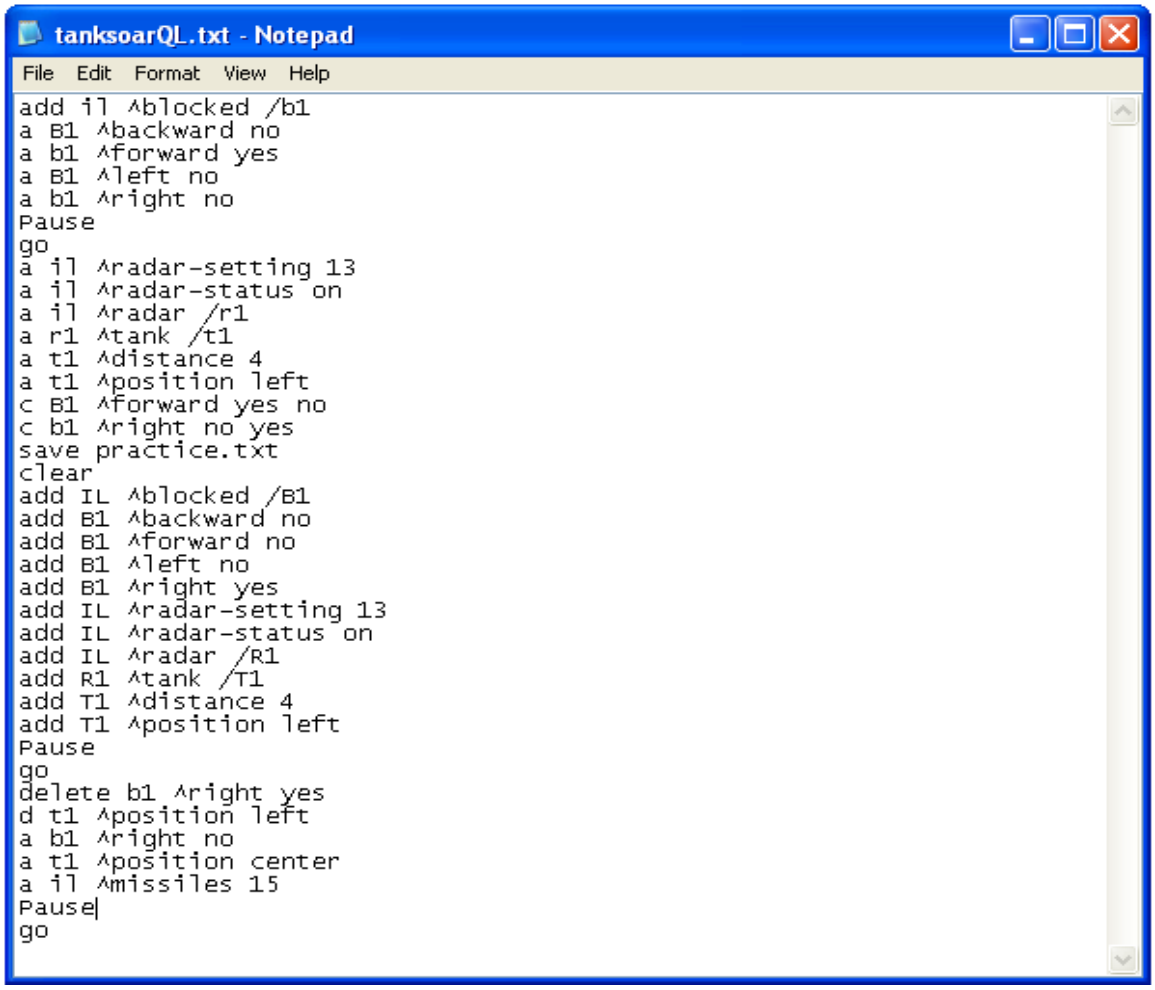
```
add il ^blocked /b1
a B1 ^backward no
a b1 ^forward yes
a B1 ^left no
a b1 ^right no
go
a il ^radar-setting 13
a il ^radar-status on
a il ^radar /r1
a r1 ^tank /t1
a t1 ^distance 4
a t1 ^position left
c B1 ^forward yes no
c b1 ^right no yes
save practice.txt
clear
add IL ^blocked /B1
add B1 ^backward no
add B1 ^forward no
add B1 ^left no
add B1 ^right yes
add IL ^radar-setting 13
add IL ^radar-status on
add IL ^radar /R1
add R1 ^tank /T1
add T1 ^distance 4
add T1 ^position left
go
delete b1 ^right yes
d t1 ^position left
a b1 ^right no
a t1 ^position center
a il ^missiles 15
go
```

Remember the first command we used was 'debug'. This command is not saved in a script by default, but can be manually added if you want it.

Notice that the save call is part of the script. Practice .txt will be recreated each time this is run.

Notice that the load practice.txt call is not included in the script. Instead, its effects (the next 11 lines) are included. This helps to create fewer dependencies between files.

Now let's say that you want to examine each of the input-link structures before they are run. To do this we are going to add `Pause` statements before each of the calls to go. Do this and your file should look like:



tanksoarQL.txt - Notepad

File Edit Format View Help

```
add il ^blocked /b1
a B1 ^backward no
a b1 ^forward yes
a B1 ^left no
a b1 ^right no
Pause
go
a il ^radar-setting 13
a il ^radar-status on
a il ^radar /r1
a r1 ^tank /t1
a t1 ^distance 4
a t1 ^position left
c B1 ^forward yes no
c b1 ^right no yes
save practice.txt
clear
add IL ^blocked /B1
add B1 ^backward no
add B1 ^forward no
add B1 ^left no
add B1 ^right yes
add IL ^radar-setting 13
add IL ^radar-status on
add IL ^radar /R1
add R1 ^tank /T1
add T1 ^distance 4
add T1 ^position left
Pause
go
delete b1 ^right yes
d t1 ^position left
a b1 ^right no
a t1 ^position center
a il ^missiles 15
Pause
go
```

DO NOT FORGET TO SAVE YOUR CHANGES TO THE FILE! After you saved the file, go back to the QL screen, type `init` and then `clear` just as before. Now type `load tanksoarQL.txt` your screen should look as follows:

```
c:\Documents and Settings\lafritay\Desktop\soar-dev\trunk\SoarSuite\SoarLibrary\bin\Quic...
***** Current Input-Link Structure *****

(IL ^blocked B1 ^missiles 15 ^radar-setting 13 ^radar-status on ^radar R1)
  (B1 ^backward no ^forward no ^left no ^right no)
  (R1 ^tank T1)
    (T1 ^distance 4 ^position center)

> init

Agent reinitialized.

> clear

***** Current Input-Link Structure *****

(IL)

> load tanksoarQL.txt

***** Current Input-Link Structure *****

(IL ^blocked B1)
  (B1 ^backward no ^forward yes ^left no ^right no)

>
```

The script is now paused directly before the first call to 'go'. At this point you could change the input-link anyway you see fit.

To continue running the script, type `continue` or just `cont`. You should see the script advance to right before the next call to 'go'. Notice also that the SoarJavaDebugger screen is also updating with the script. Type `cont` twice more to finish running the script.

6.4 Using `clears`

QL automatically begins storing input-link structures that are run from the time it is launched in case you want to save them a s a script. At any time, this memory can be cleared by typing `clears`.

Using `clears` correctly is the key to manipulating scripts. For example, how would you add a new input-link structure to the beginning of tanksoarQL.txt? The proper way to do this would be to type `clears`, enter the new input-link structure and type `go`, then run through the whole script stored in tanksoarQL.txt. Now typing `script <newfile.txt>` will save tanksoarQL.txt with a new input-link structure before it. Similarly, if you wanted to save the last portion of a certain script, you would run the script until reaching the structure you wanted to start at (using Pause), type `clears`, then after executing the last structure that is to be included in the script, you would type `script`.

7 Table of Commands

Name	Shortcut	Arguments/Examples	function
add	a	<parent id> ^<path> /<my id> add il ^blocked /B1	Adds an identifier to <parent id> with path <path> and id <my id>. Must have '/' sign. il is the default id for the input-link.
add	a	<parent id> ^<path> <value> add B1 ^forward yes	Adds a string, int or float to <parent id> with path <path> and value <value>.
change	c	<parent id> ^<path> <old value> <new value> change B1 ^forward yes no	Changes element with <parent id> and path <path> from <old value> to <new value>. <old value> and <new value> must be of same type.
clear	--	-- clear	Clears current input-link structure, has no effect on script memory.
clears	cs	-- clears	Clears out all input-link structures currently stored in script memory
debug	--	-- debug	Spawns the SoarJavaDebugger and creates a remote connection to it.
delete	d	<parent id> ^<path> /<my id> delete il ^blocked /B1	Deletes an identifier from <parent id> with path <path> and id <my id>. Must have '/' sign.
delete	d	<parent id> ^<path> <value> delete B1 ^forward no	Deletes a string, int or float from <parent id> with path <path> and value <value>.
ends	es	-- ends	Only used while running a script loaded from a file, ends the running of the script.
go	g	-- go	Runs Soar until output is generated by the Agent. This is an alias for run -o.
input	--	-- input	Re-prints the last input structure
load	l	<filename.txt>	Loads filename.txt

		load structure.txt	whether it is a script or just a single structure
local	--	--	QuickLink will create a new instance of the Soar Kernel internally
output	--	-- output	Re-prints the last output structure
remote	--	-- remote	Forces QuinkLink to destroy its internal instance of the Soar Kernel and attempts to remotely connect it to another instance.
save	s	<filename.txt> save structure.txt	Saves the current input-link structure to filename.txt.
script	sc	<filename.txt> script filename.txt	Saves all input-link structures stored in script memory as a script in filename.txt.
quit or exit	--	-- quit or exit	Exits the program