

**The Engineer's Guide to Soar
Course 01: Soar Essentials**

Project 10: Soar Markup Language (SML) Basics

By Dr. Bryan Stearns, 2024



Problem

Our agent relies on `fake_agent_input...` rules that simulate input.

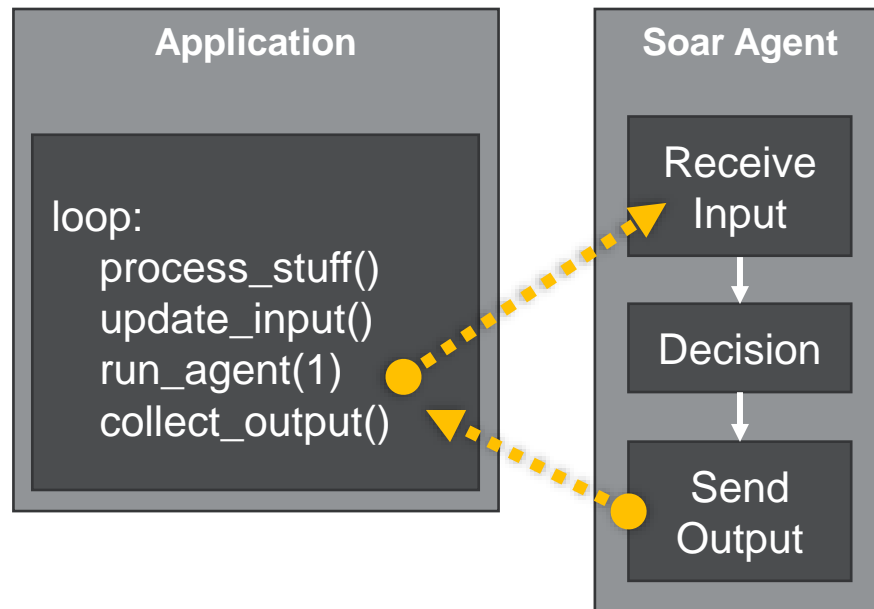
Solution

Make a Python script that runs our Soar agent and its I/O.

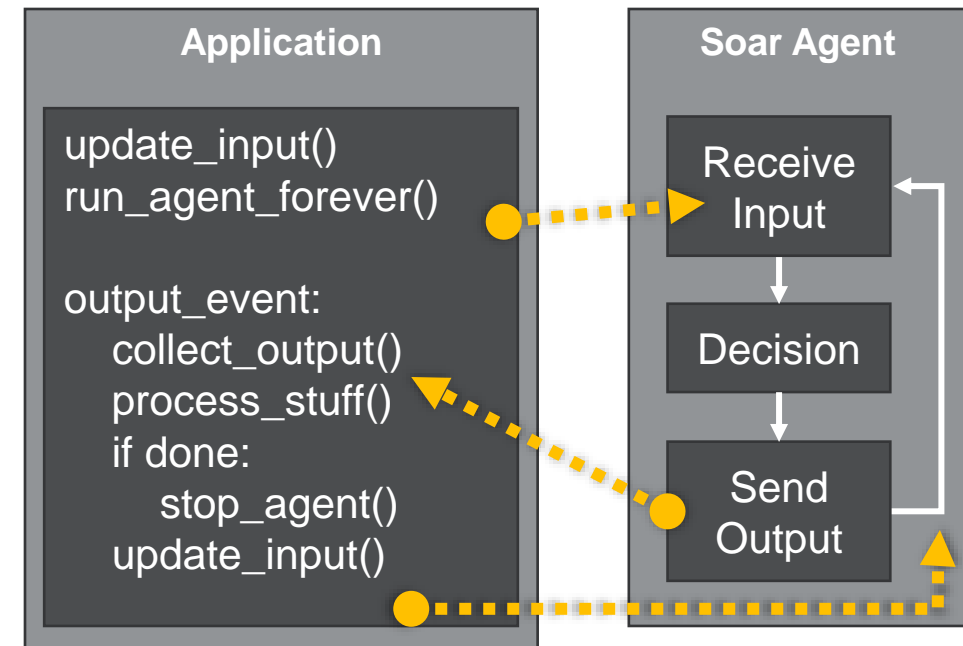
Lesson 00 Review – How Does Soar Fit Into Software?

An interfacing application calls Soar's run methods from its code
It can use event listeners if desired.

Using Direct Control



Using Event Listeners



Long-term Project Goal

Make a standalone Python app that:


- Sends inputs of our choice to our Soar agent
- Read the supplier-list output from our agent
- Can repeat this process for many input jobs given in a list

We will develop our code in 4 app iterations

- Each will represent its own mini-project and lesson
- Each app iteration will introduce new concepts
- The 4th app iteration should provide our desired final product!

Lessons 10-13 – Outline

Each of the ensuing SML lesson explains the following new concepts:



You are
here

10. App 1: SML Basics

- Kernels and Agents
- Identifier and WME classes
- Agent creation, I/O, and shutdown

11. App 2: WME Management

- Removing input WMEs properly
- Opening a linked debugger

12. App 3: I/O Management

- Using custom input classes
- Looping over output WMEs

13. App 4: Event Handlers

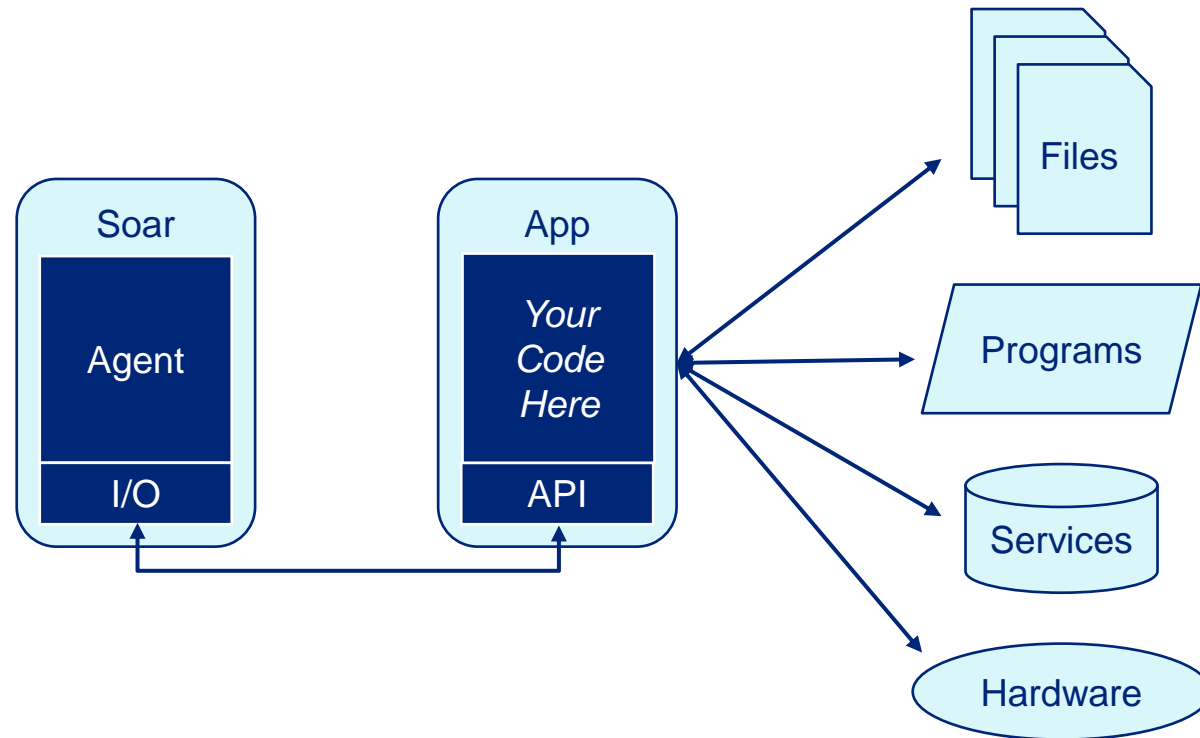
- Using event listeners
- Stopping Soar from code

SML Concepts

Connecting with Soar Kernels and Agents

Soar: Agent Environment

To connect a Soar agent to an interactive domain, you must build an app that interfaces between that domain's processing and the Soar agent's I/O.



SML: Messages vs API

The Soar process communicates with your app via SML messages

- SML = “Soar Markup Language” (a variant of XML)

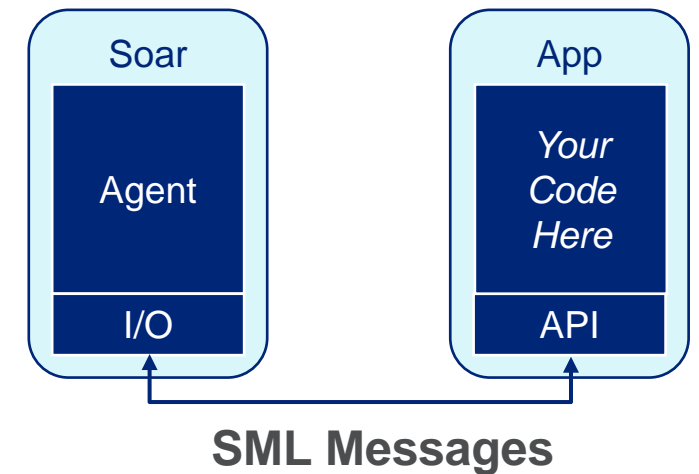
Soar comes with an API that handles SML messaging for you.

- Soar coders almost never need to interact with XML-level messages directly,
- So people usually say “SML” to refer to the API, not to the messaging language.

We will follow this convention here.

Soar provides SML libraries for C++, Java, Python, C#, and Tcl.

- (Generated when Soar’s source code is built with SWIG)

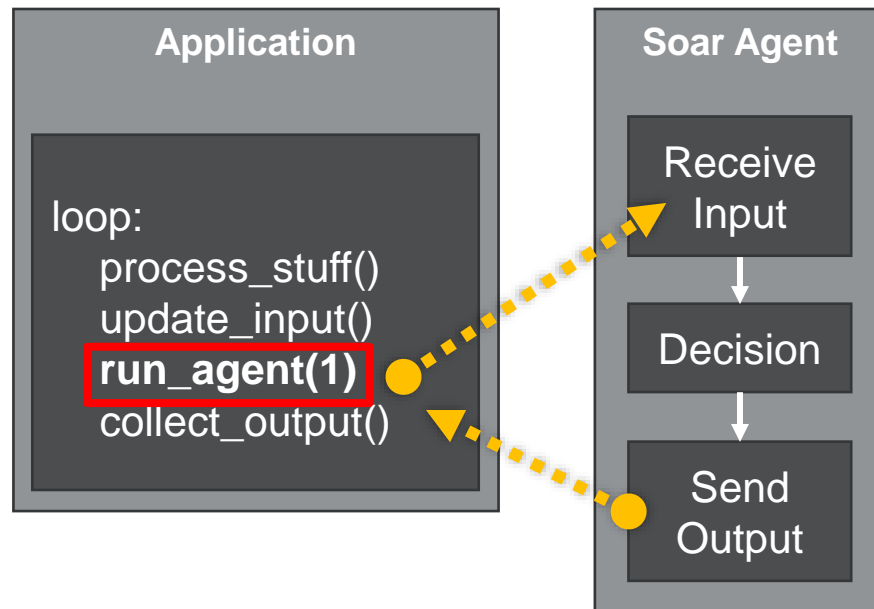


What's in the SML API?

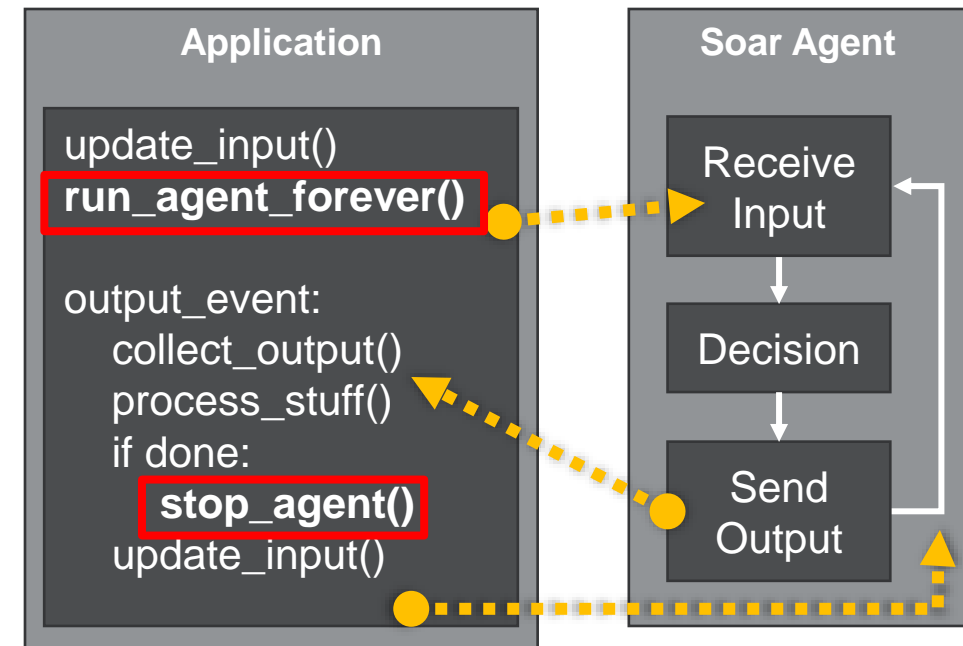
SML (the API) provides the *library of functions* that let application code:

- Create and destroy the Soar instance
- Send input and receive output
- Send any CLI command and get its result string
- And more!

Using Direct Control



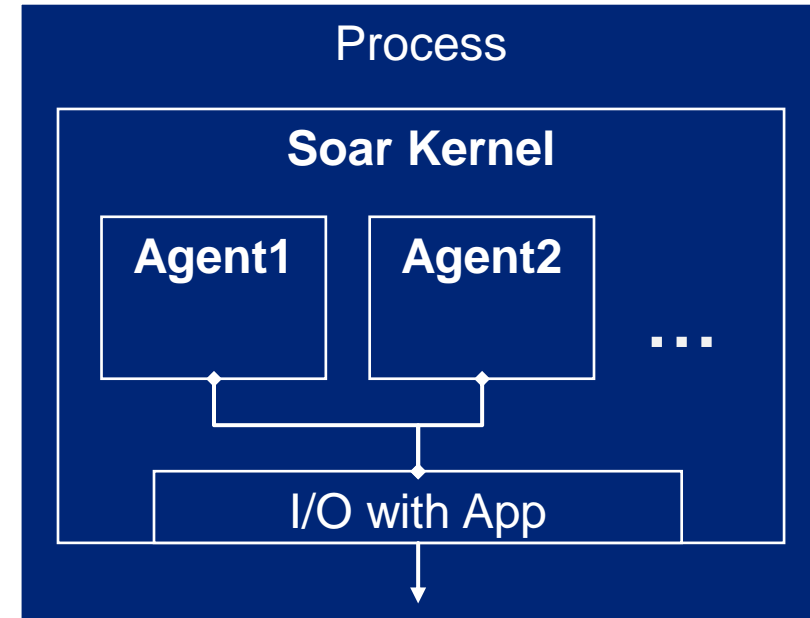
Using Event Listeners



Soar Kernels and Agents

Soar agents run inside a *Soar Kernel*:

- **Soar Kernel:** Runs the architecture
 - Soar's static functionality
 - I/O with your app code
 - Can create/maintain multiple *Soar agents*
 - Max one *kernel* per CPU process
- **Soar Agent:** Applied instance of Soar
 - Runs with agent knowledge, task state, decision cycle, etc.



Using SML to Run Soar

You can create and run a Soar agent in just a few lines of code!

1. Create a kernel instance
2. Create one or more agents in that kernel
3. Load rules for your agent(s)
4. Run the agent(s)!

```
kernel = Kernel.CreateKernelInNewThread();  
agent = kernel.CreateAgent("AgentName");  
agent.LoadProductions("file_name.soar");  
agent.RunSelfForever();
```

SML function names (like *RunSelfForever()*) are the same across all programming languages!

Closing Kernel Resources

Your code should always close a kernel when you are done with it!

- This frees up resources from it and any agents it maintains

```
kernel.Shutdown()
```

You can also free up resources from an agent but keep the kernel:

```
kernel.DestroyAgent(agent)
```

Python Import of SML

There are two ways to use Soar for Python:

1. Download the [latest Soar release](#)
 - Must manually set up env variables on your machine (see Lesson 00).
 - In Python: use **import Python_sml_ClientInterface**
2. Use [pip install soar-sml](#)
 - Automatically sets up Soar environment for your system.
 - In Python: use **import soar_sml**

This course uses the 1st option, because we can distribute the installation with this tutorial.

- It will also better equip you to use Soar in non-Python environments.
- But if starting from scratch and you only need Python, the 2nd option is much simpler. (Importing the `run_soar.py` file included in this tutorial sets up env vars for you.)

App 1

SML Basics

App 1 Goals

Our first app will be a single Python script that:

1. Creates a Soar agent in a Soar kernel
2. Loads that agent with our agent rules
3. Give the agent some input
 - For now, we'll just create a *single* candidate-supplier as input.
4. Runs the agent
 - The agent will output a list of recommended suppliers.
5. Reads the agent's output
 - For now, we'll just read the *count* of list items, and the *first supplier* in the list.
6. Shuts down Soar

Let's Write Some Code – Import SML!

First we need to import our Python SML library.

1. Open your app1/code_starter.py file
 - Notice the provided imports set up your environment variables for you
2. Find the comment labeled “STEP 1” and follow the instructions there.
 - Import the Python SML library with the following line of code:

```
import Python_sml_ClientInterface as sml
```

3. Test that the import works without errors by running the Python file
 - Nothing should happen, but you should see no error messages.

Let's Write Some Code – Create a Kernel!

Let's set up a Soar kernel instance in Python!

1. Find the comment labeled “STEP 2.1” and follow the instructions there.
 - Uncomment the “print” and “kernel = ” lines.
 - Replace the underscores with the appropriate function call:

```
kernel = sm1.Kernel.CreateKernelInNewThread(SOAR_PORT)
```

2. Find the comment at the end of the file labeled “STEP 2.2” and follow its instructions.
 - Uncomment the “print” line.
 - Close the kernel using the appropriate function call:

```
kernel.Shutdown()
```

3. Test your code by running the script.
 - You should see both your print messages on the terminal without errors.

Let's Write Some Code – Create an Agent!

Now let's create a Soar agent instance in our new kernel!


1. Find the comment labeled "STEP 3" and follow the instructions there.
 - Replace the underscores with the appropriate function call:

```
agent = kernel.CreateAgent("My Agent")
```

2. Find the comment labeled "STEP 4.1" and follow its instructions.
 - Uncomment the "print" line.
 - Load the agent rules using the appropriate function call:

```
agent.LoadProductions(AGENT_PATH)
```

3. Find the comment labeled "STEP 4.2" and follow its instructions to test your code.
 - Uncomment the "print" line.
 - Run the script. You should see the named rule printed on the terminal without errors.
 - (You can comment this line out again once you've tested it.)



**Pick any agent name.
It doesn't matter here.
(Names act as unique
identifiers if you have
>1 agent in a kernel.)**

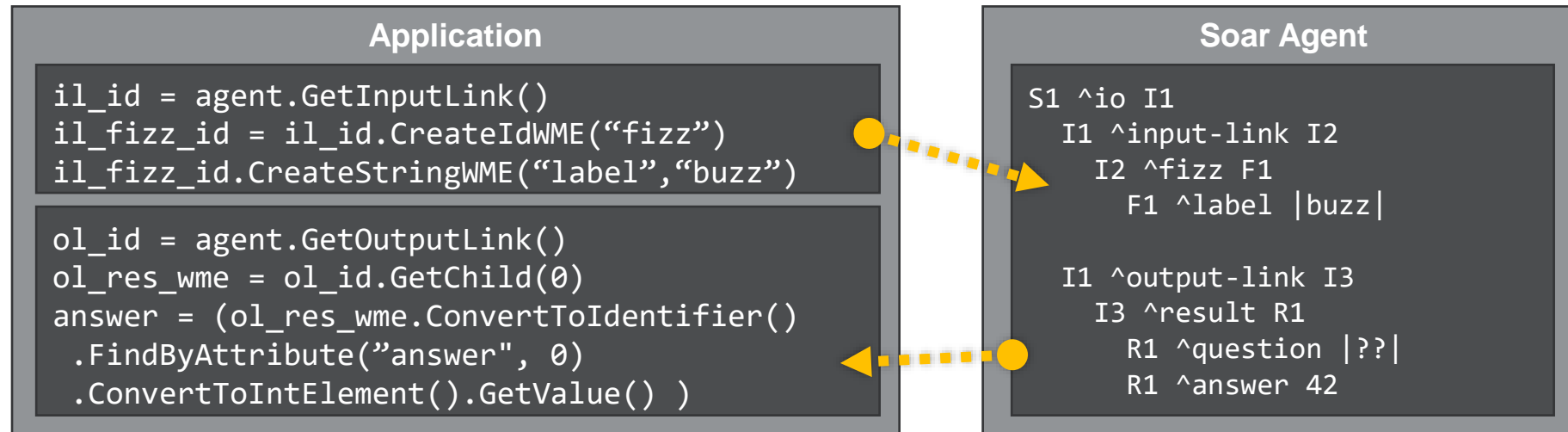
Interacting with Agent WMEs using SML

To manage agent I/O, we access its `^input-link` and `^output-link` WMEs.

- Then we can add/remove/read WMEs underneath them!

SML code maintains a parallel copy of the agent's WMEs that it has references for.

- Reading WMEs in code pulls the updates from the agent's WM as needed.
- After SML code modifies the WMEs, changes are committed to the agent's WM
 - You can optionally turn off auto-commit behavior and control when commits are sent.



SML Classes

You use two SML classes in particular when navigating/modifying a Soar graph from SML:

1. **WMElement**

- An object that represents a single WME
- Lets you access any of the WME's ID, Attribute, or Value data

2. **Identifier**

- An object that represents a single ID / memory object
- Lets you access any child WMEs / augmentations under that ID

There are also classes for the different Value data types:

- IntElement, FloatElement, StringElement, IdentifierSymbol

WMElement is a parent class to all these other classes

- To get a WME's Value in its appropriate type, you must convert the WMElement to the right class
 - (See the "ConvertTo..." functions in the example on the previous slide.)


Let's Write Some Code – Send Soar Input

Now let's use the input-link!

1. Find the comment labeled “STEP 5” and follow the instructions there.
 - Uncomment the code and replace the underscores as follows to get the input-link ID:

```
il_id = agent.GetInputLink()
```

2. Find the comment labeled “STEP 6” and follow its instructions.
 - Uncomment the code and replace the underscores appropriately.
 - You want to create the following structure on the input-link:
 - The comment instructions describe the functions you need.
 - First use `il_id` to create the candidate-supplier WME.
 - Then use the returned Identifier to create the remaining WMEs.

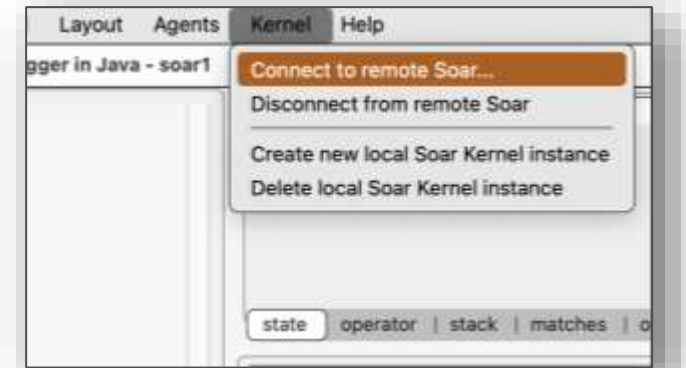


```
I1 ^input-link I2
I2 ^candidate-supplier C1
C1 ^name |supplier01|
C1 ^total-cost 24.99
C1 ^total-yeses 2
```

Connecting the Debugger to the Kernel

We can connect a Java Debugger window to inspect the agent we created with SML

1. Put a break point before you shut down your kernel
2. Run your SML script in *Debugging* mode
3. Open a Soar Java Debugger window
 - Run the SoarJavaDebugger.sh/.bat file (See Lesson 00.)
4. From the Debugger, navigate to *Kernel* > *Connect to remote Soar*
5. Click Ok to shut down the kernel that was created for the Debugger when it launched.
6. Change the Port number to 12345 to match your code.
 - Leave the IP address blank to indicate a local connection.
7. You should now be connected to the kernel run by your paused script!
 - Run it for one step and inspect the input-link to see your WMEs.



Be Mindful of Ports!

When you open the Debugger, it creates its own kernel that uses the default port 12121.

- It will use a different port if 12121 is already claimed.

If you run your script and it tries to use the same port that the Debugger already claimed, you will get an error:

```
Creating Soar Kernel using port 12121
Error: Error binding the listener socket to its port number
Failed to create the internet listener socket. Shutting down listener thread.
```

This error won't prevent your app from running your kernel locally, but it will prevent other apps (like the Debugger) from connecting to your kernel via an internet socket:port.

Try it. Change your script to use port 12121 and run it *after* opening the Debugger.

Let's Write Some Code – Run the Agent

We could run the agent using the Debugger after we connect it to our agent...

But we want to run the agent from SML!

1. Find the comment labeled “STEP 7” and follow the instructions there.

- Uncomment the print statement
- Add this function call to run the agent until something stops it:

```
agent.RunSelfForever()
```

2. Test it by seeing the agent run on its own in the Debugger

- Put a breakpoint on this new line in your SML code
- Run the app in debug mode up to this line
- Open a Soar Debugger window and connect it to your agent
- Step over the `agent.RunSelfForever()` line to execute it
- See the Soar Debugger run the agent to completion!

***Recall that our agent
will interrupt itself
when it sends output!***

```
*****
*****
Total: 32 productions sourced.
  1: 0: 01 (init)
  2: 0: 02 (select-supplier)
** OUTPUT: First try supplier01
  3: 0: 03 (output-supplier-list)
*** DONE ***
Interrupt received.
```


Let's Write Some Code – Reading the Soar Output-Link

Now let's get our app to read the output created by our Soar agent!

1. Find the comment labeled “STEP 8” and follow the instructions there.

- Uncomment just these lines:
- Replace the underscores with this to get the output-link ID:

```
ol_id = agent.GetOutputLink()
```

```
# ol_id = ____  
# if ol_id is None:  
# print("No output detected yet.")
```

2. Run your script

- You should see the “No output detected yet.” message. What happened?
- `GetOutputLink()` will return `None` until the agent has sent its first output.
- But if you check the Soar Debugger, you'll notice that your agent *has* sent output. So why does `ol_id == None`?
- Our agent interrupted itself as soon as it generated its output. That means it also stopped Soar from transmitting its output-link updates!

Let's Write Some Code – Run a Single Step

We need our agent to run just a little longer

1. Go back to the comment labeled “STEP 9” and follow the instructions there.

- Use the following to run the agent for a single decision cycle:

```
agent.RunSelf(1)
```

2. Now run your script again

- You should **not** see the “No output detected yet.” message anymore!
- This means that `ol_id` will now let you read the agent's output-link.

Now we just need to read the WMEs that are on this output-link ID....

Reading the Agent's Output

We just want to read 2 parts of the agent's output for now:

- The count of items in the output recommendation list
- The name of the first supplier in the recommendation list (if the count is >0)

Recall that the agent's output is structured as follows:

```
^output-link ID
  ^supplier-list ID      # The ID of a single supplier list
    ^supplier ID        # The ID of a supplier object in the list
      ^name STRING      # The name of the supplier
      ^rank INT          # The position of this supplier in the list
      ^next ID           # The next supplier in the list
    ^first-supplier ID   # The ID of the first supplier in the list
    ^last-supplier ID    # The ID of the last supplier in the list
    ^count INT           # The number of suppliers in the list
```

So we'll want to access:


- `output-link.supplier-list.count INT`
- `output-link.supplier-list.first-supplier.name STRING`

Finding a WME By Attribute

SML provides the `<ID>.FindByAttribute(attribute, N)` function

- It iterates through the child WMEs of the calling Identifier object and returns the Nth one that matches the given attribute name.
- N is 0-indexed, so setting N=0 returns the first matching WME
- If there is no match, the function returns None.

For example: `o1_id.FindByAttribute("supplier-list",0)`
will return this WME:



```
^output-link I3
  ^supplier-list C1
    ^supplier C2
      ^name |supplier01|
      ^rank 1
    ^first-supplier C2
    ^last-supplier C2
    ^count 1
```

But if you want to then find a WME connected under the result, you must use `<WME>.ConvertToIdentifier()` to get the Identifier that is the *Value* of the WME. (C1 in the example to the right.)

You can then use `FindByAttribute(...)` again to get that Identifier's children.

Getting a WME's Value

To get an ID Value from a WME, use:

```
<WME>.ConvertToIdentifier()
```

--> returns *Identifier* object

To get a String Value from a WME, use:

```
<WME>.ConvertToStringElement()
```

--> returns *StringElement* object

To get an Integer Value from a WME, use:

```
<WME>.ConvertToIntElement()
```

--> returns *IntElement* object

To get a Float Value from a WME, use:

```
<WME>.ConvertToFloatElement()
```

--> returns *FloatElement* object

You can also always get the String form of a Value without converting first using:

```
<WME>.GetValueAsString()
```

--> returns *str*

For String, Integer, or Float, to get the literal value, use:

```
<String/Int/Float-Element>.GetValue()
```

--> returns *str/int/float*

Let's Write Some Code – Read the Agent's Output

Uncomment everything from the “else” through to the end of the try/except block.

1. STEP 10: Read the list count

- Replace the underscores to set `list_count` appropriately.
- You can do this in one line by chaining `<ID>.FindByAttribute(...)` and `<WME>.ConvertToIdentifier()`.
- Convert the count to an `IntElement` before getting the integer value.
- Running the script should then print: “List size is 1.”

2. STEP 11: Read the first supplier's name

- Replace the underscores to set `first_chan_name` appropriately.
- The pattern of code is similar to STEP 10, but you can use `GetValueAsString()` at the end instead of `ConvertToIntElement().GetValue()`.
- Running the script should then print “First supplier: **supplier01**”

```
else:
    try:
        # STEP 10: Read the size
        # --> Use <ID>.FindByAttribute(...)
        # --> Use <WME>.ConvertTo
        # --> Use <WME>.ConvertTo
        list_count =                 
        print(f"List size is {lis

        # STEP 11: If the count i
        # --> Use <WME>.GetValueA
        if list_count > 0:
            first_chan_name =                 
            print(f"First supplie

    except AttributeError:
        # If FindByAttribute() do
        # So catch any error resu
        print("ERROR IN OUTPUT HA
```

Exception Handling

Any time we read a WME path that might not exist, we should check for an `AttributeError`.

- This will be raised if `FindByAttribute()` returns `None`, and then we try to call a `ConvertTo...` on that `None`

```
except AttributeError:  
    # If FindByAttribute() doesn't find anything, it  
    # So catch any error resulting from output missing  
    print("ERROR IN OUTPUT HANDLER")
```

Congratulations!

You now have an SML app that can

- Send input to Soar
- Read output from Soar!

```
Running Soar!  
List size is 1.  
First supplier: supplier01  
  
Goodnight Soar.
```

Check out the next lesson to learn more about how to change input values over time on a single agent!