

TinyTSC Documentation

Latest: 20061030

Background

This is a tiny prototype from which TSC4J will evolved. It represents the smallest possible body of code that satisfies the core architectural requirements of TSC. At this time, it imports legacy TSC (old Forth-based) code, and exports to its own XML format <note>not implemented yet</note>.

See **Envisionment Trace** below for a glimpse of the XML format in tinyTSC.

Architecture

<td>

Knowledgebase

Right now, we are just using the old TSC KBs from earlier experiments. They have been cleaned up (sortof) and are found in the directory `/legacy/test` .

TODO: wire import and export so we can move to an internal XML format for future work.

Installation

Just configure the directory `/tinytsc` according to the following directory structure. There are three jar files currently required in the `/lib` directory:

- `jdbm.jar`
- `log4j.jar`
- `xpp3.jar`

The system includes a database engine (JDBM¹) that builds a database in the `/database` directory. At this time, the system does nothing directly with that database. It will come into use once we extend this development platform to include mappings from other sources of knowledge structures (frames).

Directory Structure

```
/root
  /tinytsc
    /classes
    /doc
    /database
    /images
    /legacy (raw data)
    /lib
```

¹JDBM: <http://jdbm.sourceforge.net/> behaves essentially as a persistent hashmap.

```
/logs
/src
build.xml
logfile.log
todo.txt
tsc-props.xml
```

build.xml includes several *ant* targets, for compiling, building a jar file, building the javadocs, and running tinyTSC.

Configuration

The TSC directory structure includes the file tsc-props.xml. In that file is an important property:

```
<parameter name="DatabasePath" value="/database" />
```

DatabasePath is presently set to the *base relative* path /database. At this time, TSC is hard-wired to prepend that value with the path to the install directory, /tinytsc . A future change will add a new property: PathIsAbsolute with "1" = true and "0" = false. This would allow for installation of the database data files anywhere desired on the same computer.

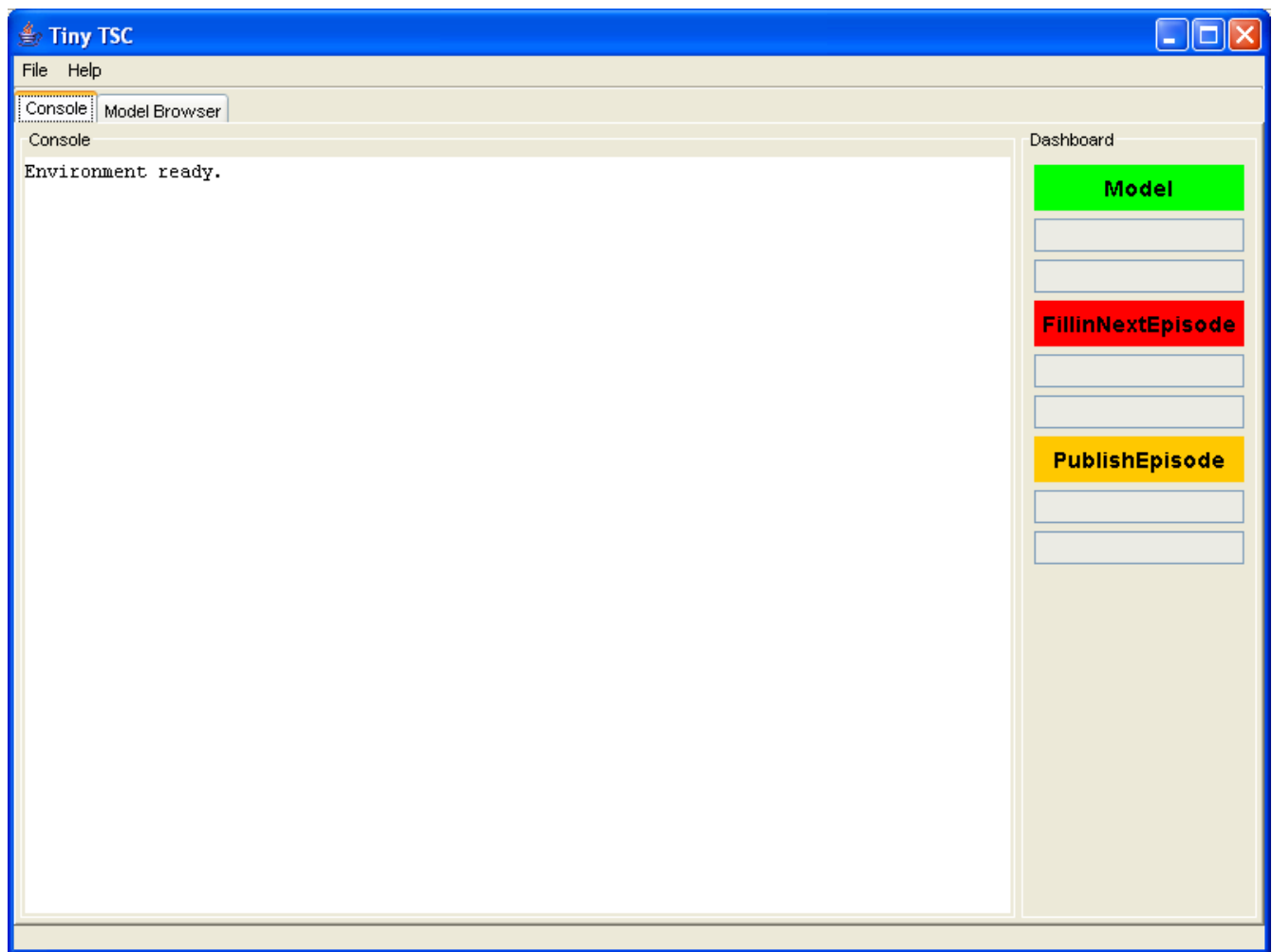
Running

Several ways to run tinyTSC. From a shell script, you need to link in the jars in /lib, then call org.nex.tinytsc.Main

If %ANT_HOME% is configured, simply call from /tinytsc directory

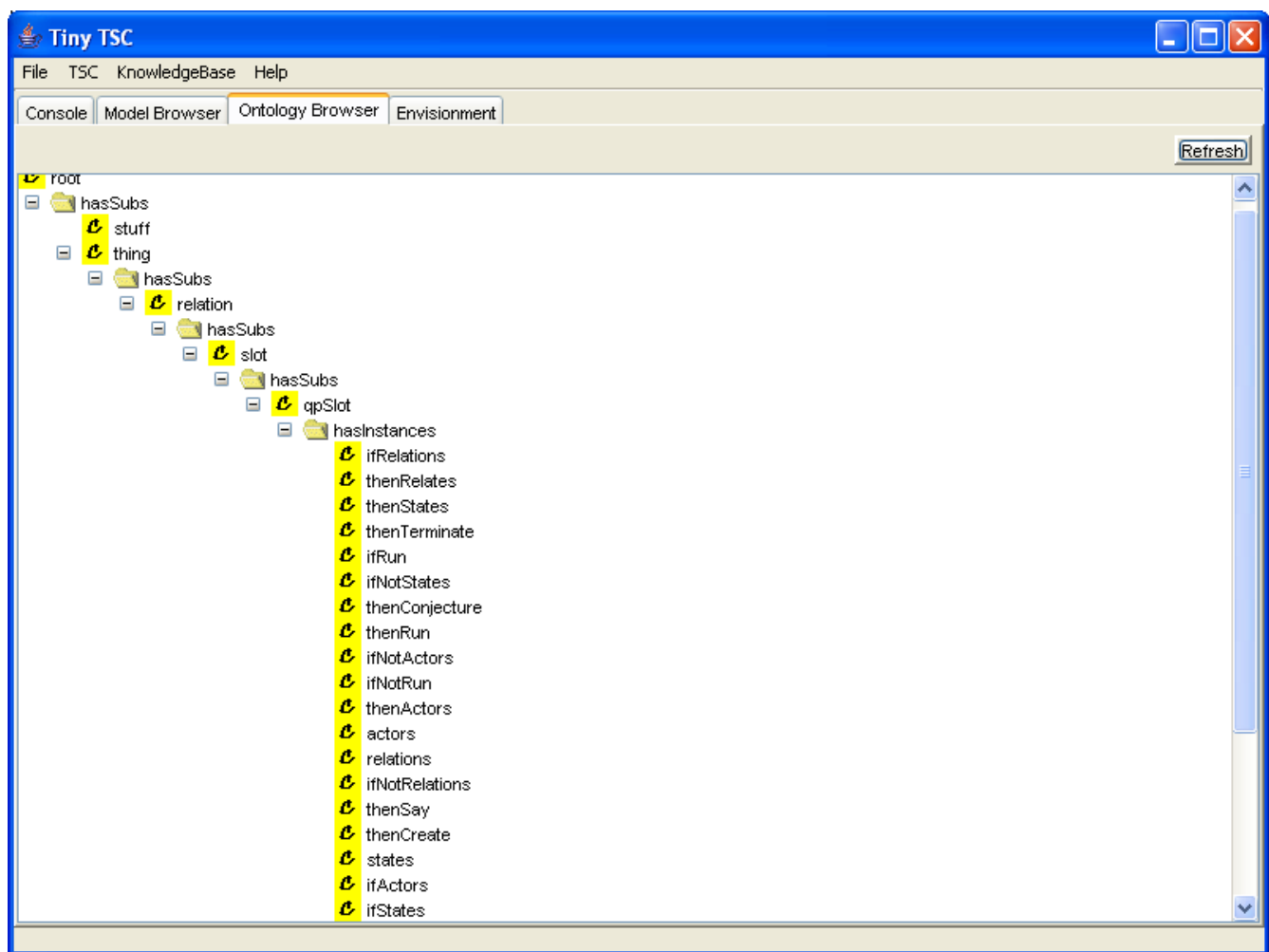
```
ant run
```

There are a number of shortcuts taken in this code, one of which is that the code expects tsc-props.xml to be found in the install directory from where the classpath originates: /tinytsc.

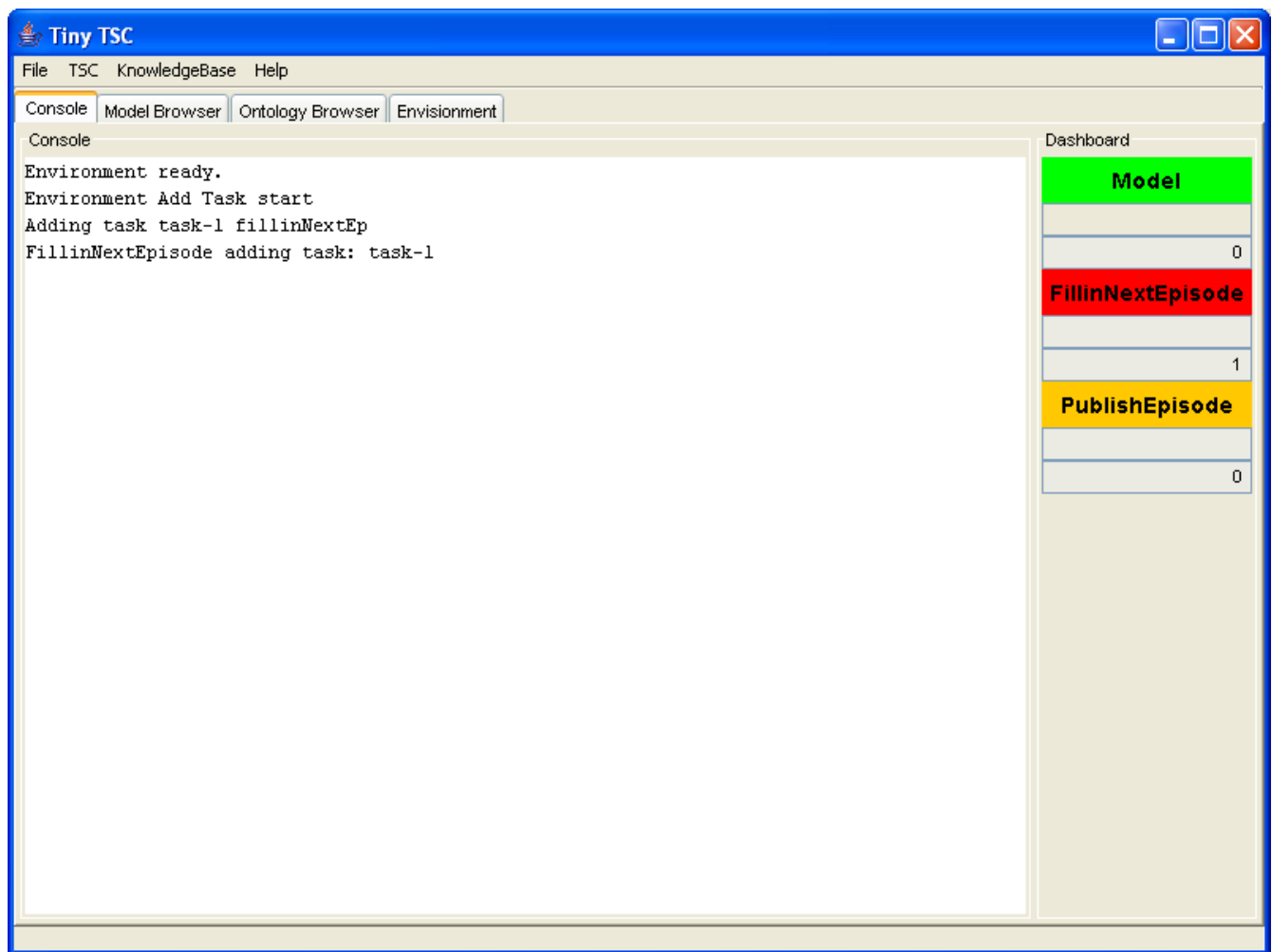


On Bootup

Using: `File:Import Legacy`, then navigate to `/legacy/test/_testLoader` results in the following imported Ontology.

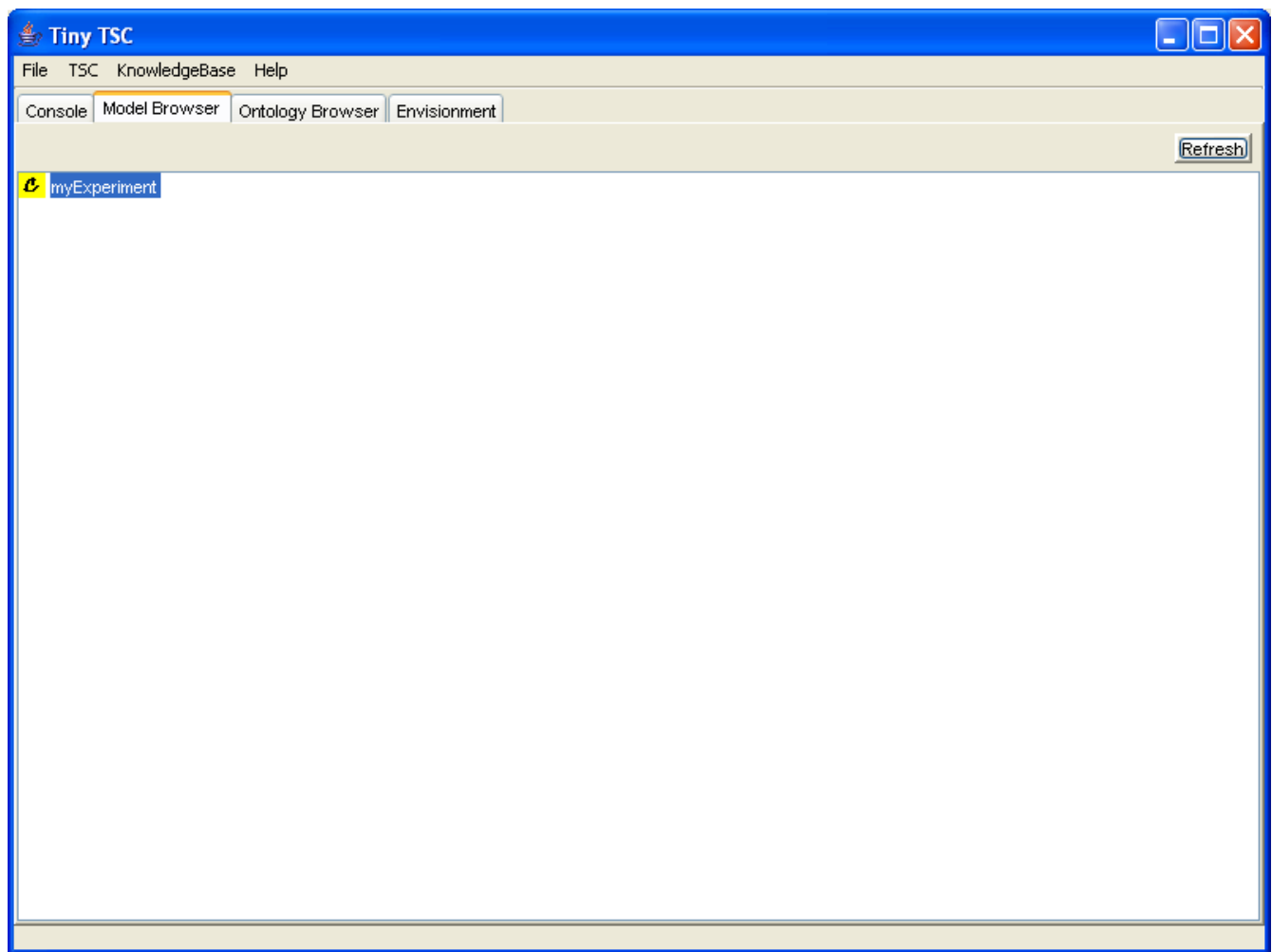


After Importing Legacy



Console after Importing Legacy

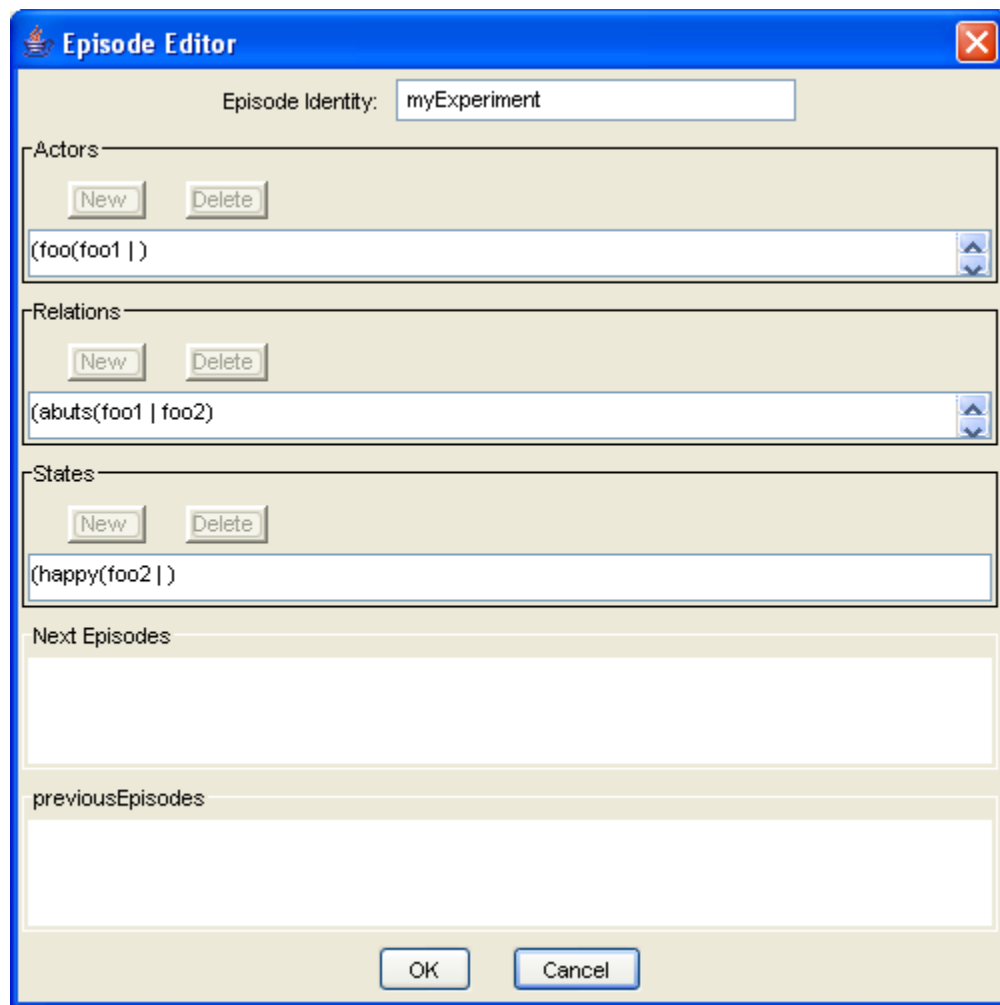
Select the Model Browser tab and observe the empty model. Double click on it and observe the Episode Editor.



Empty Model after Loading

<note>This tab should, after running tasks, show the full envisionment as a tree of nodes. Right now, it fails to do so</note>

<note>Ontology Browser tab should, after running tasks, show envisionments below their models. Not showing. Perhaps due to only class/subclass links being painted</note>



The image shows a software dialog box titled "Episode Editor". At the top, there is a label "Episode Identity:" followed by a text input field containing "myExperiment". Below this, the dialog is divided into three main sections: "Actors", "Relations", and "States". Each section has a "New" button and a "Delete" button. The "Actors" section contains a text field with the value "(foo(foo1 |))". The "Relations" section contains a text field with the value "(abuts(foo1 | foo2))". The "States" section contains a text field with the value "(happy(foo2 |))". Below these sections are two empty list boxes labeled "Next Episodes" and "previousEpisodes". At the bottom of the dialog are "OK" and "Cancel" buttons.

Episode Identity: myExperiment

Actors

New Delete

(foo(foo1 |))

Relations

New Delete

(abuts(foo1 | foo2))

States

New Delete

(happy(foo2 |))

Next Episodes

previousEpisodes

OK Cancel

Model Browser with Initial Conditions

Using TSC:Run Task... the system will exercise the imported tasks, all those installed in the Agenda. That results in the following Console trace.

The screenshot shows the Tiny TSC application window. The top menu bar includes 'File', 'TSC', 'KnowledgeBase', and 'Help'. Below the menu is a tabbed interface with 'Console', 'Model Browser', 'Ontology Browser', and 'Envisionment'. The 'Console' tab is active, displaying a log of system events. The 'Dashboard' panel on the right shows a hierarchical view of the model structure.

Console Trace:

```

Environment ready.
Environment Add Task start
Adding task task-1 fillinNextEp
FillinNextEpisode adding task: task-1
Starting tasks
Starting: publishEp
Starting: fillinNextEp
Setting currentModel: myExperiment
FillinNextEpisode on: myExperiment
FillinNextEpisode got 2 predicates.
FillinNextEpisode got 3 rules.
FillinNextEpisode filtering on predicates: [foo, bar]
FillinNextEpisode checking rule preds: [foo, bar]
FillinNextEpisode checking rule preds: [foo, bar]
FillinNextEpisode checking rule preds: [foo, bar]
FillinNextEpisode filtered 3 rules.
FillinNextEpisode testing 3 rules
FillinNextEpisode pushing bindings
FillinNextEpisode testing rule rule-1
FillinNextEpisode testing rule rule-1 on episode myExperiment
FillinNextEpisode tested rule rule-1 on episode myExperiment : true
FillinNextEpisode firing rule rule-1 on episode myExperiment
Rule 1 fired
FillinNextEpisode new publish task on E1161823721671
FillinNextEpisode newTask T1161823721671
Environment Add Task start
Adding task T1161823721671 publishEp
PublishEpisode adding task: T1161823721671

```

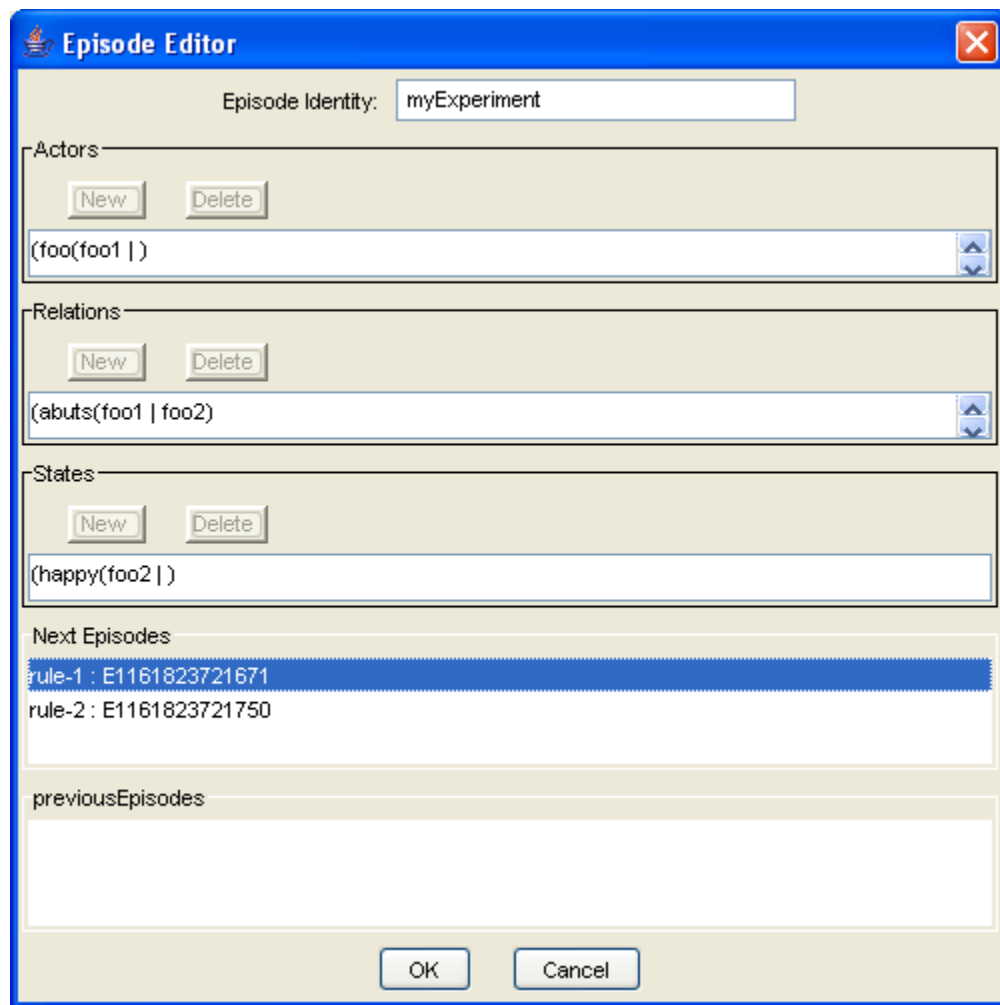
Dashboard:

Model	
myExperiment	6
FillinNextEpisode	
E1161823721671	1
PublishEpisode	
E1161823721796	0

Console Trace after running Tasks

This also results in a new view in the Model Browser when double clicking the first episode.

<note>some bug in the code (nullpointerexception at `org.nex.tinytsc.engine.Episode.compareEpisode(Episode.java:222)`) seems to prevent the display of the two generated episodes, but they do appear in the Episode Editor as follows</note>



The image shows a software window titled "Episode Editor" with a blue title bar and a red close button. The window contains several sections for editing an episode. At the top, there is a text field labeled "Episode Identity:" with the value "myExperiment". Below this are three sections: "Actors", "Relations", and "States". Each section has a "New" and a "Delete" button, followed by a text field. The "Actors" field contains "(foo(foo1 |))", the "Relations" field contains "(abuts(foo1 | foo2))", and the "States" field contains "(happy(foo2 |))". Below these is a "Next Episodes" section with a list box containing two items: "rule-1 : E1161823721671" (which is selected with a blue background) and "rule-2 : E1161823721750". At the bottom is a "previousEpisodes" section with an empty list box. The window ends with "OK" and "Cancel" buttons.

Episode Identity: myExperiment

Actors

New Delete

(foo(foo1 |))

Relations

New Delete

(abuts(foo1 | foo2))

States

New Delete

(happy(foo2 |))

Next Episodes

rule-1 : E1161823721671

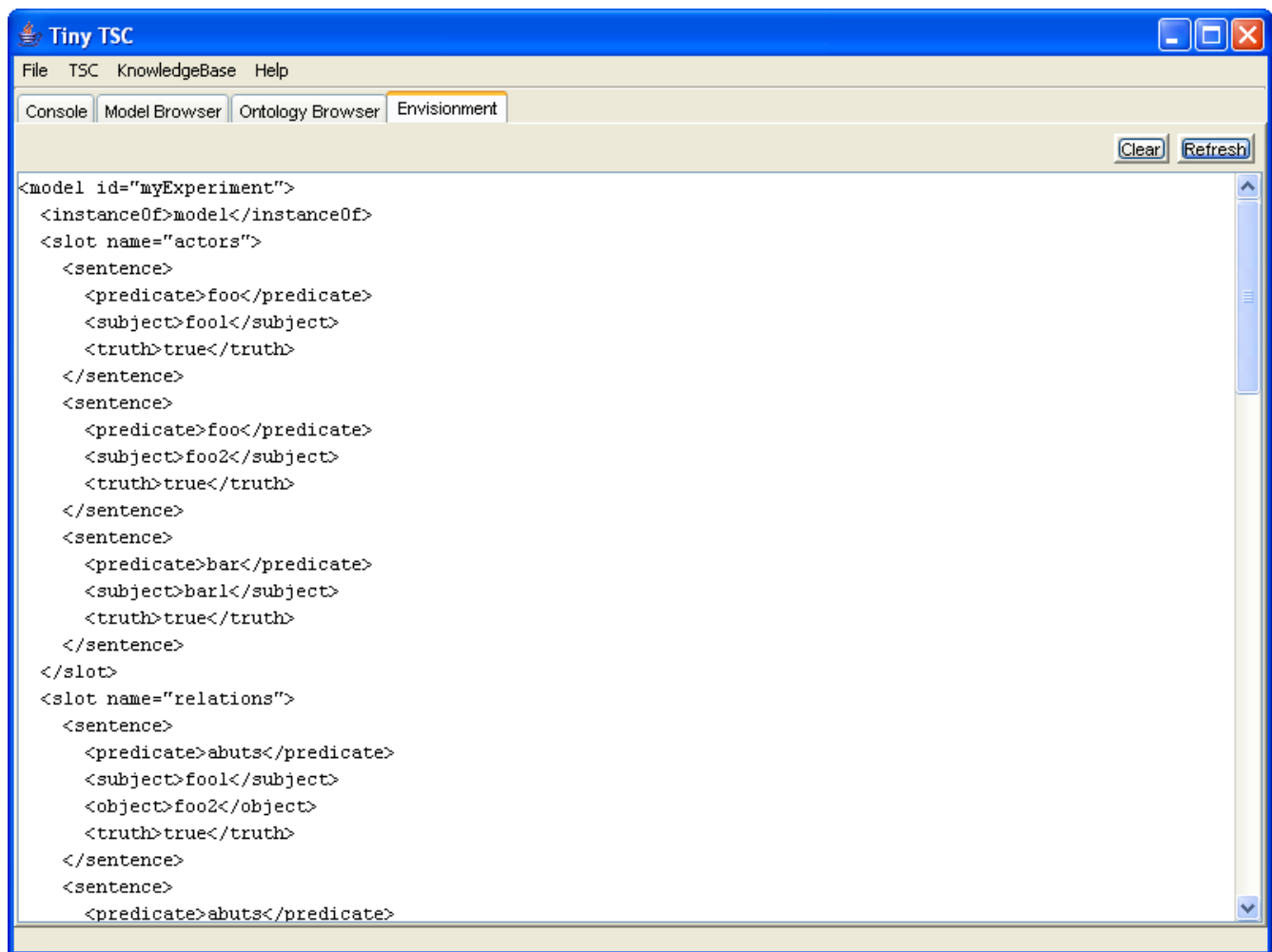
rule-2 : E1161823721750

previousEpisodes

OK Cancel

Episode Editor on Initial Conditions showing 2 Next Episodes

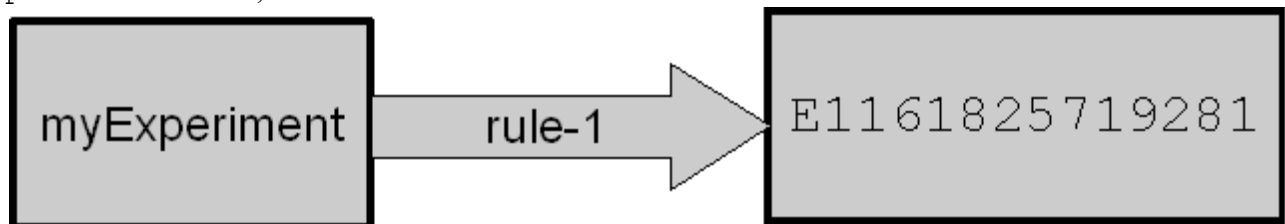
Click Refresh on the Envisionment tab to see the XML trace of the result of a task firing.



XML trace of the Environment

Environment Trace

.A model is taken as the initial conditions for an environment. It is then followed by episodes. As such, it is the root of a tree.



```
<model id="myExperiment">
  <instanceOf>model</instanceOf>
  <slot name="actors">
    <sentence>
      <predicate>foo</predicate>
      <subject>fool</subject>
      <truth>true</truth>

```

```

    </sentence>
    <sentence>
      <predicate>foo</predicate>
      <subject>foo2</subject>
      <truth>true</truth>
    </sentence>
    <sentence>
      <predicate>bar</predicate>
      <subject>bar1</subject>
      <truth>true</truth>
    </sentence>
  </slot>
  <slot name="relations">
    <sentence>
      <predicate>abuts</predicate>
      <subject>foo1</subject>
      <object>foo2</object>
      <truth>true</truth>
    </sentence>
    <sentence>
      <predicate>abuts</predicate>
      <subject>foo2</subject>
      <object>bar1</object>
      <truth>true</truth>
    </sentence>
  </slot>
  <slot name="states">
    <sentence>
      <predicate>happy</predicate>
      <subject>foo2</subject>
      <truth>true</truth>
    </sentence>
  </slot>
  <nextEpisode>
    <rule>rule-1</rule>
    <node>E1161825719281</node>
  </nextEpisode>
</model>

<episode id="E1161825719281">
  <instanceOf>episode</instanceOf>
  <myMechanism>rule-1</myMechanism>
  <slot name="actors">
    <sentence>
      <predicate>foo</predicate>
      <subject>foo1</subject>
      <truth>true</truth>
    </sentence>
    <sentence>

```

```
    <predicate>foo</predicate>
    <subject>foo2</subject>
    <truth>true</truth>
  </sentence>
  <sentence>
    <predicate>bar</predicate>
    <subject>bar1</subject>
    <truth>true</truth>
  </sentence>
</slot>
<slot name="relations">
  <sentence>
    <predicate>abuts</predicate>
    <subject>fool</subject>
    <object>foo2</object>
    <truth>true</truth>
  </sentence>
  <sentence>
    <predicate>abuts</predicate>
    <subject>foo2</subject>
    <object>bar1</object>
    <truth>true</truth>
  </sentence>
</slot>
<slot name="states">
  <sentence>
    <predicate>sad</predicate>
    <subject>fool</subject>
    <truth>true</truth>
  </sentence>
  <sentence>
    <predicate>happy</predicate>
    <subject>foo2</subject>
    <truth>true</truth>
  </sentence>
</slot>
<previousEpisode>
  <rule>rule-1</rule>
  <node>myExperiment</node>
</previousEpisode>
</episode>
<note>missing </model></note>
```