LEGEND User Guide

PoC: Robert P. Goldman
mailto:rpgoldman@sift.net

2025/10/14

This document describes SIFT's system, LEGEND, a knowledge acquisition (KA) tool for a subset of Ergo/Rulelog. LEGEND is a web application: users will access the tool through dedicated URLs during the initial period of experimental use. LEGEND supports KA aimed at generating two kinds of Ergo content: (1) Rules, currently limited to a predicate dialect of Rulelog (see below) and (2) Frame-based specifications of instances of known types.

The version delivered 2025/10/15 is an MWS and has a number of limitations that we describe in this document.

1 Accessing LEGEND

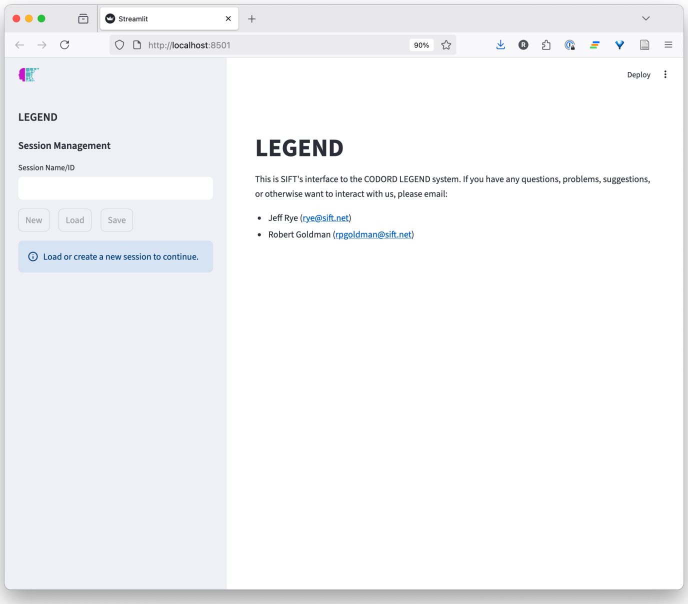
Users will be given individual URLs to reach LEGEND. These URLs should *generally* not be shared, but a single KA team may find it advantageous to share a single URL in order to share a single, consistent model. In the future it may be possible to provide collaboration support. URLs will be supplied separately.

2 Data Preparation

Background knowledge Because LEGEND is a web application, background knowledge files must be modified before they can be added to the application. The application does not have access to the user's filesystem, so constructs like `#include` directives will not work correctly.

1. **All include directives must be removed** from the files before upload. Instead of using include directives, upload the files to be included together with the files that include them.

2. Loading files into modules does not work properly at the moment. To the best of our knowledge, the only way to get Ergo code into a particular module is through the file loading interface – there is no way for a file to mark itself as destined for a particular module. For now, **all the background knowledge files must be loaded into the same module**.



LEGEND

Session Management

Session Name/ID

Load or create a new session to continue.

LEGEND

This is SIFT's interface to the CODORD LEGEND system. If you have any questions, problems, suggestions, or otherwise want to interact with us, please email:

- Jeff Rye (rye@sift.net)
- Robert Goldman (rpgoldman@sift.net)

Figure 1: Initial LEGEND screen.

> This module is not specified by the user: it will be controlled by LEGEND. However, it will be possible to load the rules derived from KA into any module of the user's choice for deployment.

Rule and Frame prompts As Benjamin pointed out at the PI meeting, it is helpful to extract, and possibly rephrase, statements describing rules and frame instances from running text ahead of time. This will speed interaction with LEGEND.

Delivery package Included as an attachment to the LEGEND delivery package is a set of modified files for the two example cases we have received, together with example prompts extracted from text.

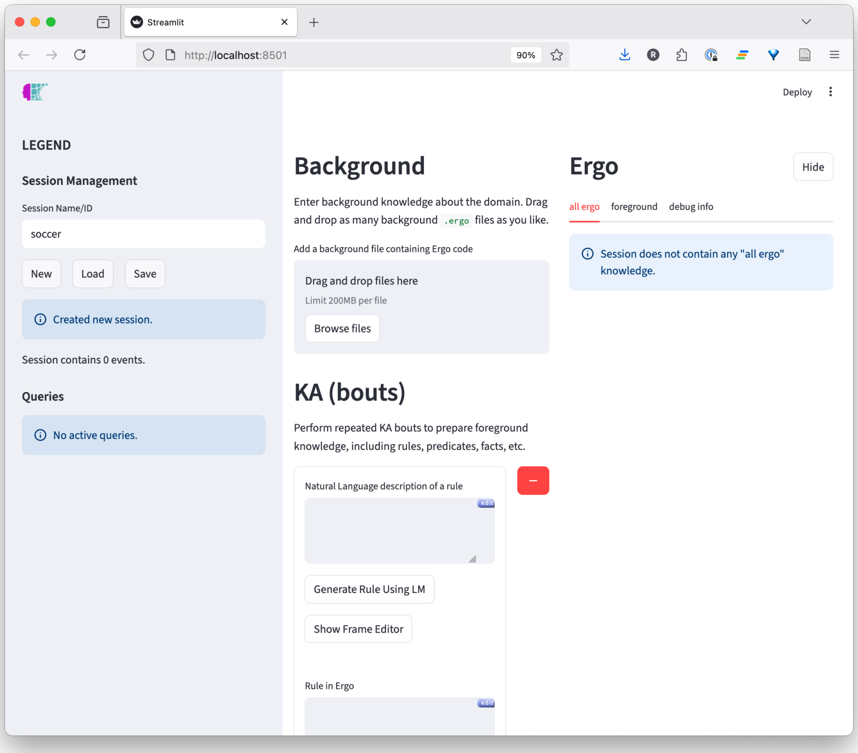
3 Opening the application

On startup, the application will open a browser window, which will show only a message and the session configuration screen (Figure 1). The application window will be largely blank until one creates or reloads a session.

4 Starting a Session

The KA *session* is the organizing container for KA on a particular subject. To create a session, type a session name into the textbox in the left tab, and press the "New" button.

After creating a session, LEGEND will reconfigure to a "triptych" display (Figure 2). The leftmost pane contains session management controls, the middle pane is where the KA takes place, and the rightmost shows the Ergo code that is available, both "background" (given) and "foreground" (the results of KA).



Created new session.

Session contains 0 events.

No active queries.

Enter background knowledge about the domain. Drag and drop as many background `.ergo` files as you like.

Add a background file containing Ergo code

Drag and drop files here

Limit 200MB per file

Perform repeated KA bouts to prepare foreground knowledge, including rules, predicates, facts, etc.

Natural Language description of a rule

Session does not contain any "all ergo" knowledge.

Figure 2: LEGEND screen with empty session.

5 Adding background knowledge

The next step in starting KA is to supply any required background (given) knowledge. To do this, you may either drag and drop files into the upload area or use a file picker. See Figure 3, which shows the file "`soccer_team_background.ergo`" loaded. You may load multiple background files, as needed.

The right pane of the window will update to show the uploaded background knowledge (this pane can be collapsed to allow more space for KA and session control).

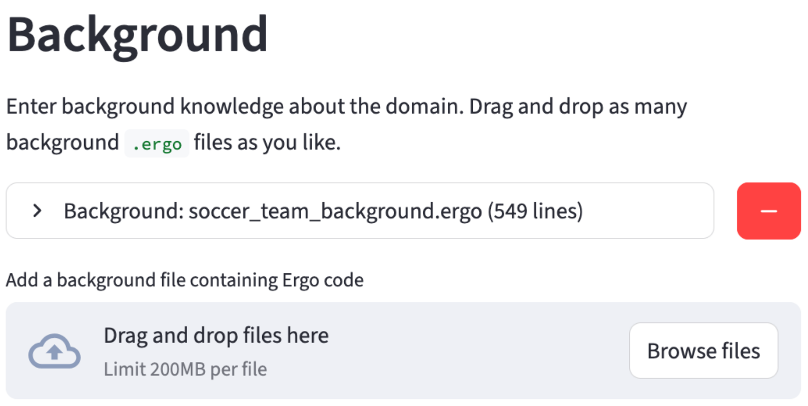
Files can be deleted from the set of background knowledge using the minus sign in the bright red box. Again, the right pane display will update accordingly.

Unfortunately, users cannot *update* background knowledge files, because of the limitations of a web application (the web application has no access to the user's file system), so all one can do in that case is delete a file and re-upload it.

After successfully uploading the background knowledge you want, it's a good idea to save the session. At present, the session will be saved on the server. You can reload it by name.

6 KA in bouts

KA in LEGEND proceeds in what we are calling "bouts." Each bout centers around a piece of textual input, which you paste into a text box (Figure 4), and then proceeds either to rule authoring (Section 7) or frame population (Section 8).


Background

Enter background knowledge about the domain. Drag and drop as many background `.ergo` files as you like.

Background: soccer_team_background.ergo (549 lines)

Add a background file containing Ergo code

Figure 3: Background file selection.

Note: "bout" may not have been the best term for us to use, since it implies strict ordering. One can enter the NL for a bout, realize that it's not one you are ready to work on, and postpone KA in favor of another. For example, if populating a frame instance, you might realize that you need another frame instance as a property filler. That's ok! Start a new bout and come back to this one.

Warning: if you save a session and restart, incomplete bouts may not properly be restored. We are working on this.

7 KA for Rules

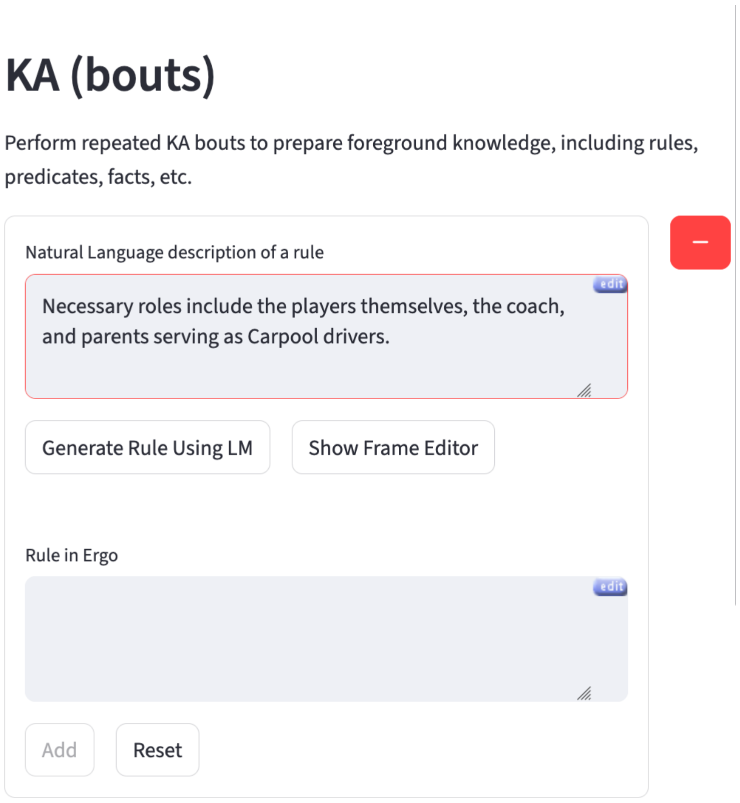
Figure 4 gives an example of a rule to be added to the foreground knowledge. To get suggested rule(s) press the "Generate Rule using LM" button. LEGEND will assemble a prompt from the input NL and information from the background knowledge, and submit it to a LLM (right now we are using Anthropic's Claude).

The result, for the "necessary roles" prompt can be seen in Figure 5. LEGEND will display the chain of thought output from the language model and, separately, the Ergo rule(s).

At this point, you can edit the ergo rule, modify the NL input and retry, or add the rule to the current set of foreground knowledge. We will soon be adding the opportunity to instruct the LM to modify its proposed rule. That capability already exists in LEGEND, but has not yet been connected to the user interface.

Note that any time you are satisfied with KA products, it is a good idea to save the session, and download the products, as well (see Section 11). LEGEND will be under constant development, attempting to add new features and fix bugs, over the course of the next two weeks, so save early and often! Sometimes innocent repairs to the system will render previous sessions impossible to reload.

This would also be a good time to (re)run stored queries (see Section 10), to see if adding the rule(s) has the desired effect.


KA (bouts)

Perform repeated KA bouts to prepare foreground knowledge, including rules, predicates, facts, etc.

Natural Language description of a rule

Necessary roles include the players themselves, the coach, and parents serving as Carpool drivers.

Figure 4: Entering the NL prompt for a bout.

8 KA for Frame Instances

Figure 6 shows what happens if you enter NL and ask for the Frame editor, instead of rule generation. In the current version of the interface, you have choose the frame type yourself. We will soon connect suggestions generated by the LM: that will be particularly helpful for the test evaluation problem, which has a much larger set of types from which to choose. As with the rule generator, the frame editor displays an Ergo translation of the KA result, and this translation can be edited in place before adding the rule to the foreground knowledge.

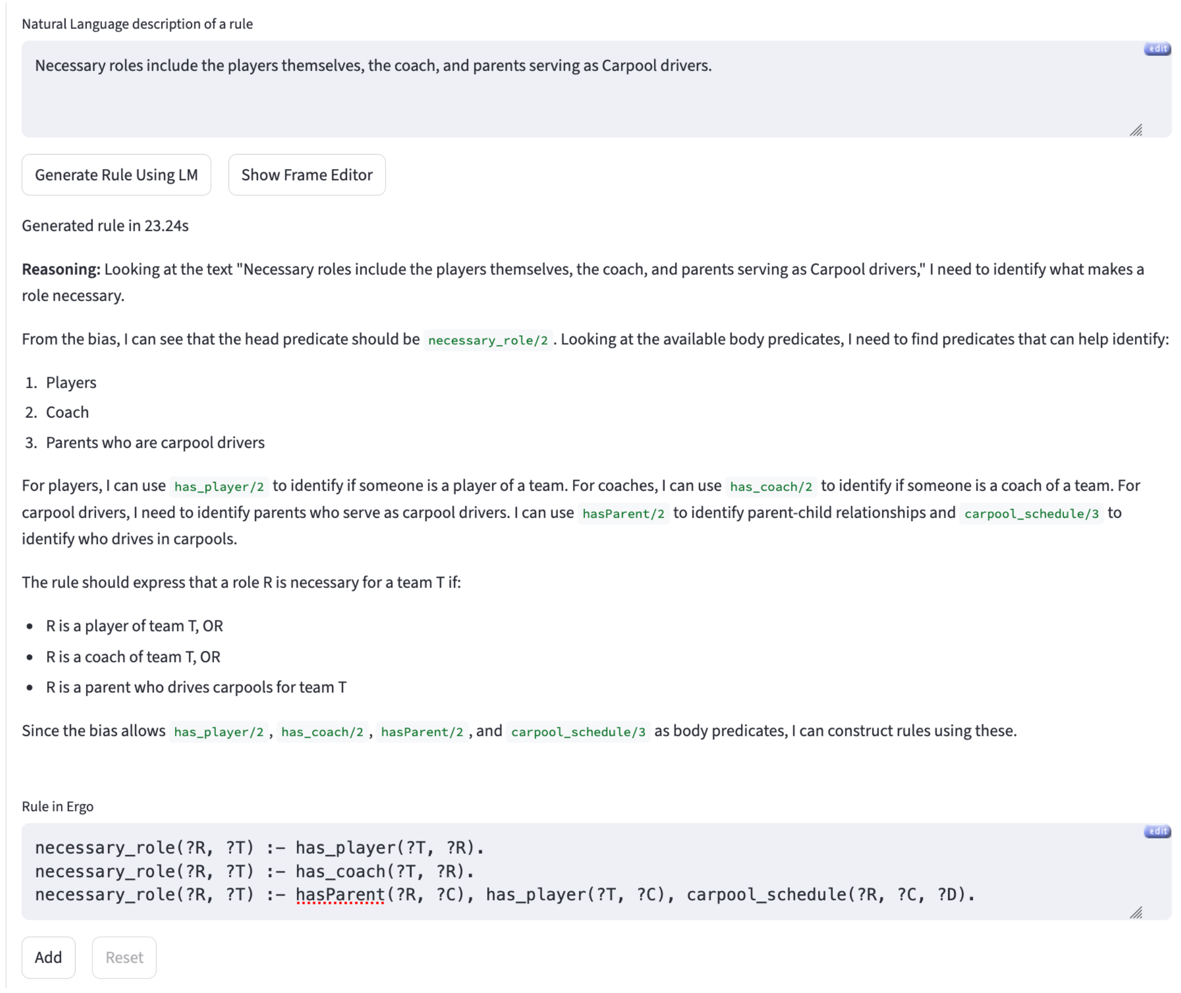
This figure shows an example of interleaving bouts. We would like to be able to record the fact that Bill, Jr. is Bill French's son, but we haven't added Bill, Jr. to the model yet. We can add a new sentence and create Bill Jr, as shown in Figure 7.

After we have added Bill, Jr. we can go back to the previous bout, and select him as Bill French's son. See Figure 8.

As pointed out in the previous section, this would be a good time to save the session, and possibly the foreground Ergo code, and also to run any stored queries.

9 KA results

At any time, you can inspect the background knowledge and KA products that have been accepted. For example, see Figure 9.


Natural Language description of a rule

Necessary roles include the players themselves, the coach, and parents serving as Carpool drivers.

Generated rule in 23.24s

Reasoning: Looking at the text "Necessary roles include the players themselves, the coach, and parents serving as Carpool drivers," I need to identify what makes a role necessary.

From the bias, I can see that the head predicate should be `necessary_role/2`. Looking at the available body predicates, I need to find predicates that can help identify:

1. Players
2. Coach
3. Parents who are carpool drivers

For players, I can use `has_player/2` to identify if someone is a player of a team. For coaches, I can use `has_coach/2` to identify if someone is a coach of a team. For carpool drivers, I need to identify parents who serve as carpool drivers. I can use `hasParent/2` to identify parent-child relationships and `carpool_schedule/3` to identify who drives in carpools.

The rule should express that a role R is necessary for a team T if:

- R is a player of team T, OR
- R is a coach of team T, OR
- R is a parent who drives carpools for team T

Since the bias allows `has_player/2`, `has_coach/2`, `hasParent/2`, and `carpool_schedule/3` as body predicates, I can construct rules using these.

Rule in Ergo

```
necessary_role(?R, ?T) :- has_player(?T, ?R).
necessary_role(?R, ?T) :- has_coach(?T, ?R).
necessary_role(?R, ?T) :- hasParent(?R, ?C), has_player(?T, ?C), carpool_schedule(?R, ?C, ?D).
```

Figure 5: Rule produced by LEGEND.

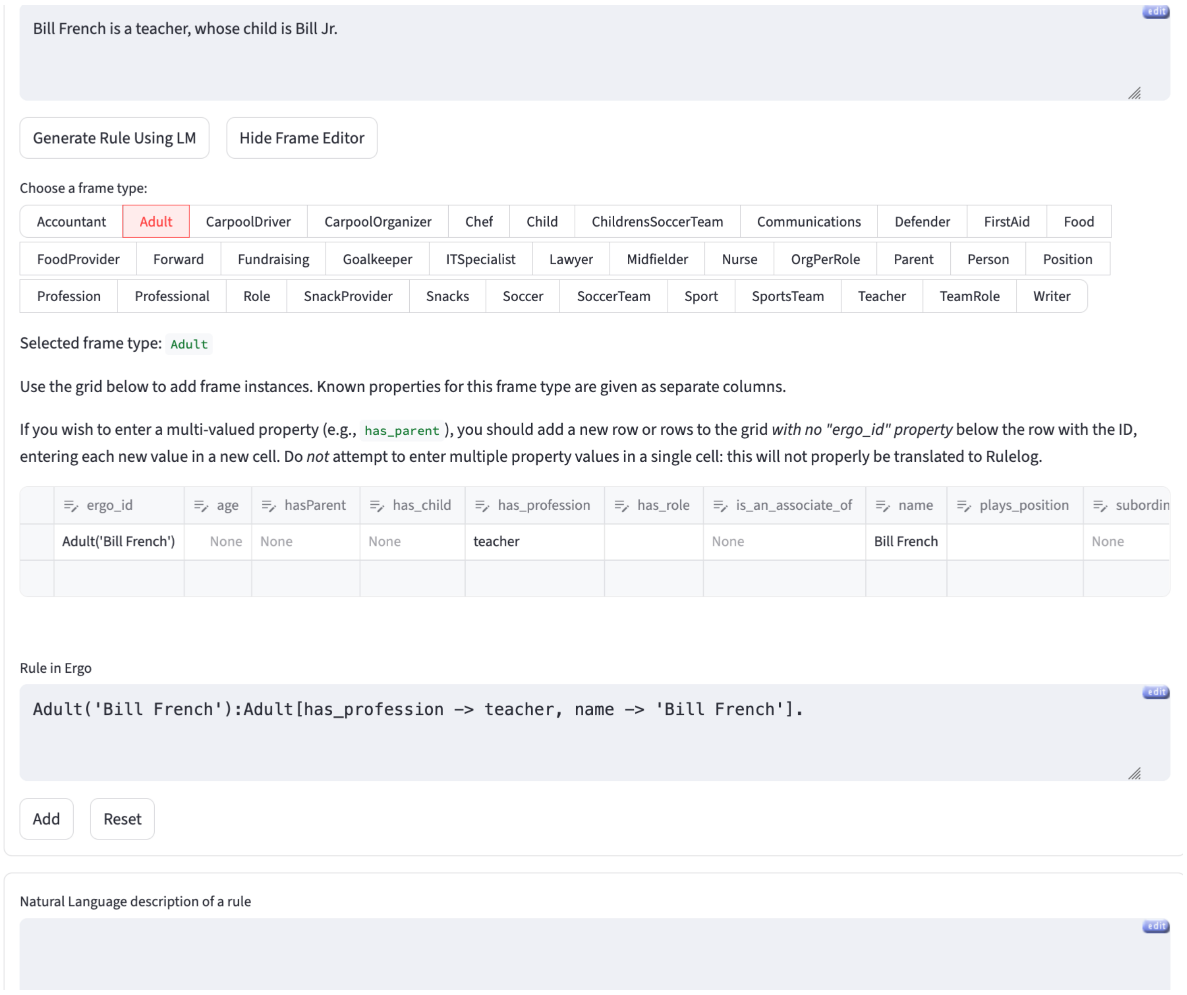
10 Queries

To help users determine whether the KA has been successful, LEGEND accommodates stored queries that can be tested against the combination of background and foreground knowledge. For example, if working on the Soccer exercise, one might want to run the `%test_scenario`. In Figure 10, we see a user adding this top-level query to LEGEND, and calling it `test all`. As this figure shows, queries are added in the middle pane of the LEGEND interface.

Queries added to the session can be run from the *left* pane, as shown in Figure 11.

11 Downloading KA products

Finally, it is possible to download either the full set of Ergo code from LEGEND, or just the newly-learned, foreground knowledge. This is done using the right pane, as shown in Figure 12. Of course, this can be done as many times as you would like.


Bill French is a teacher, whose child is Bill Jr.

Selected frame type: `Adult`

Use the grid below to add frame instances. Known properties for this frame type are given as separate columns.

If you wish to enter a multi-valued property (e.g., `has_parent`), you should add a new row or rows to the grid *with no "ergo_id" property* below the row with the ID, entering each new value in a new cell. Do *not* attempt to enter multiple property values in a single cell: this will not properly be translated to Rulelog.

	ergo_id	age	hasParent	has_child	has_profession	has_role	is_an_associate_of	name	plays_position	subordin
	Adult('Bill French')	None	None	None	teacher		None	Bill French		None

Rule in Ergo

```
Adult('Bill French'):Adult[has_profession -> teacher, name -> 'Bill French'].
```

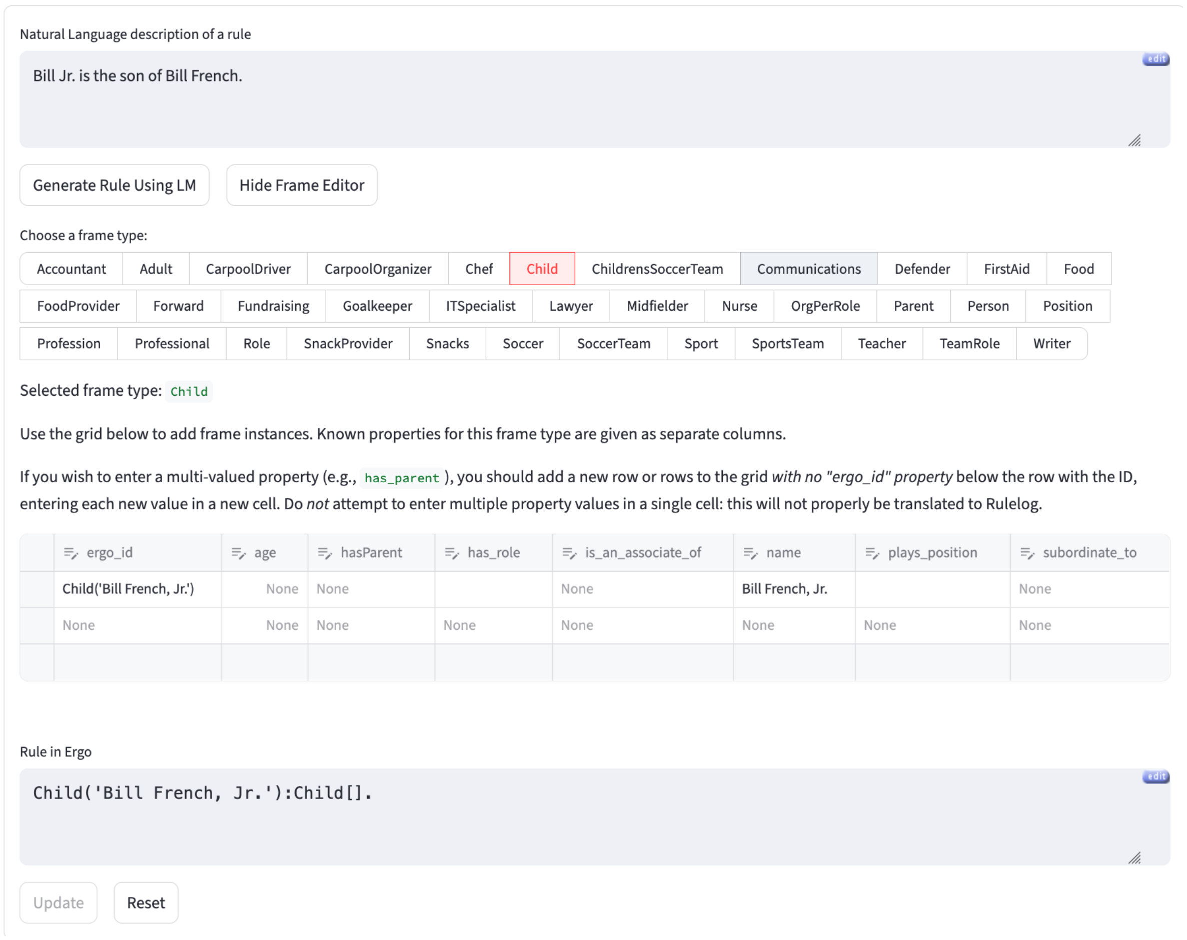
Natural Language description of a rule

Figure 6: Adding a new frame instance.

12 Lessons learned

While LEGEND is not a *conventional* IDE, it is nevertheless an IDE for the Ergo language. For this reason, it is somewhat awkward to use as a web application. The problem is that programming is still very much tied to use of a filesystem, and as a web application, LEGEND is denied access to the local filesystem. This makes it awkward to work with code that references pathnames. This limitation is particularly acute for the Ergo programming language, since Ergo relies on include directives to ensure that the code in multiple files can be loaded into a coherent process image, and allocating code to different modules is inherently tied to file loading: in Ergo files are assigned to namespaces when they are loaded. Ergo does not allow the programmer to assign a namespace to their code in the file itself, unlike many other programming languages.

There are offsetting advantages to the deploying LEGEND as a web application at this stage of development, however. For one thing, in this early stage of development, it is a great advantage to be able to rapidly incorporate fixes into the web application, without the need for an update delivery process. We can also collect information about the system's use that will guide software maintenance and enhancement. These advantages currently overwhelm


Natural Language description of a rule

Bill Jr. is the son of Bill French.

Choose a frame type:

Selected frame type: `Child`

Use the grid below to add frame instances. Known properties for this frame type are given as separate columns.

If you wish to enter a multi-valued property (e.g., `has_parent`), you should add a new row or rows to the grid *with no "ergo_id" property* below the row with the ID, entering each new value in a new cell. Do *not* attempt to enter multiple property values in a single cell: this will not properly be translated to Rulelog.

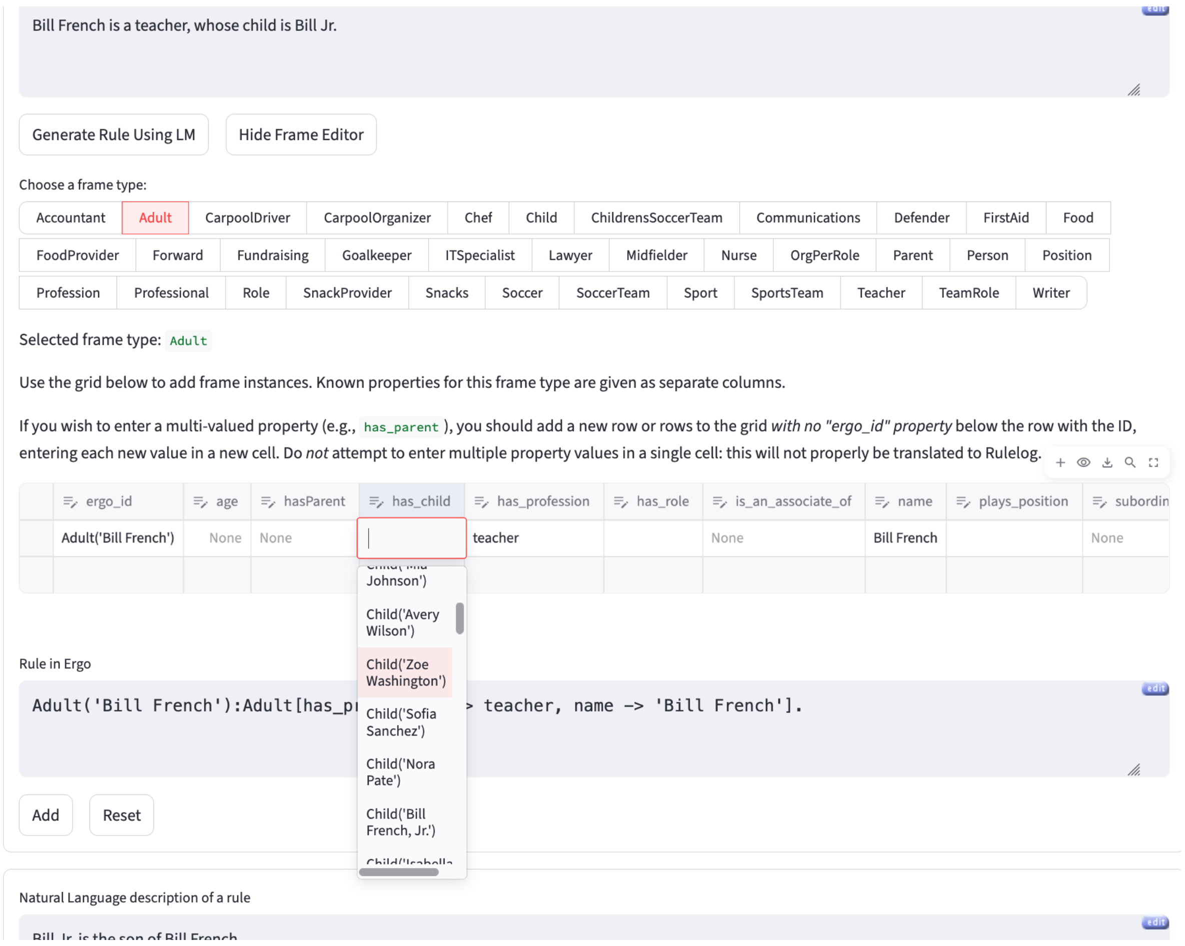
ergo_id	age	hasParent	has_role	is_an_associate_of	name	plays_position	subordinate_to
Child('Bill French, Jr.')	None	None		None	Bill French, Jr.		None
None	None	None	None	None	None	None	None

Rule in Ergo

```
Child('Bill French, Jr.'):Child[].
```

Figure 7: Adding Bill Jr.

the disadvantages, but eventually we would like to see LEGEND move to being a desktop application.


Bill French is a teacher, whose child is Bill Jr.

Selected frame type: `Adult`

Use the grid below to add frame instances. Known properties for this frame type are given as separate columns.

If you wish to enter a multi-valued property (e.g., `has_parent`), you should add a new row or rows to the grid *with no "ergo_id" property* below the row with the ID, entering each new value in a new cell. Do *not* attempt to enter multiple property values in a single cell: this will not properly be translated to Rulelog.

ergo_id	age	hasParent	has_child	has_profession	has_role	is_an_associate_of	name	plays_position	subordin
Adult('Bill French')	None	None		teacher		None	Bill French		None

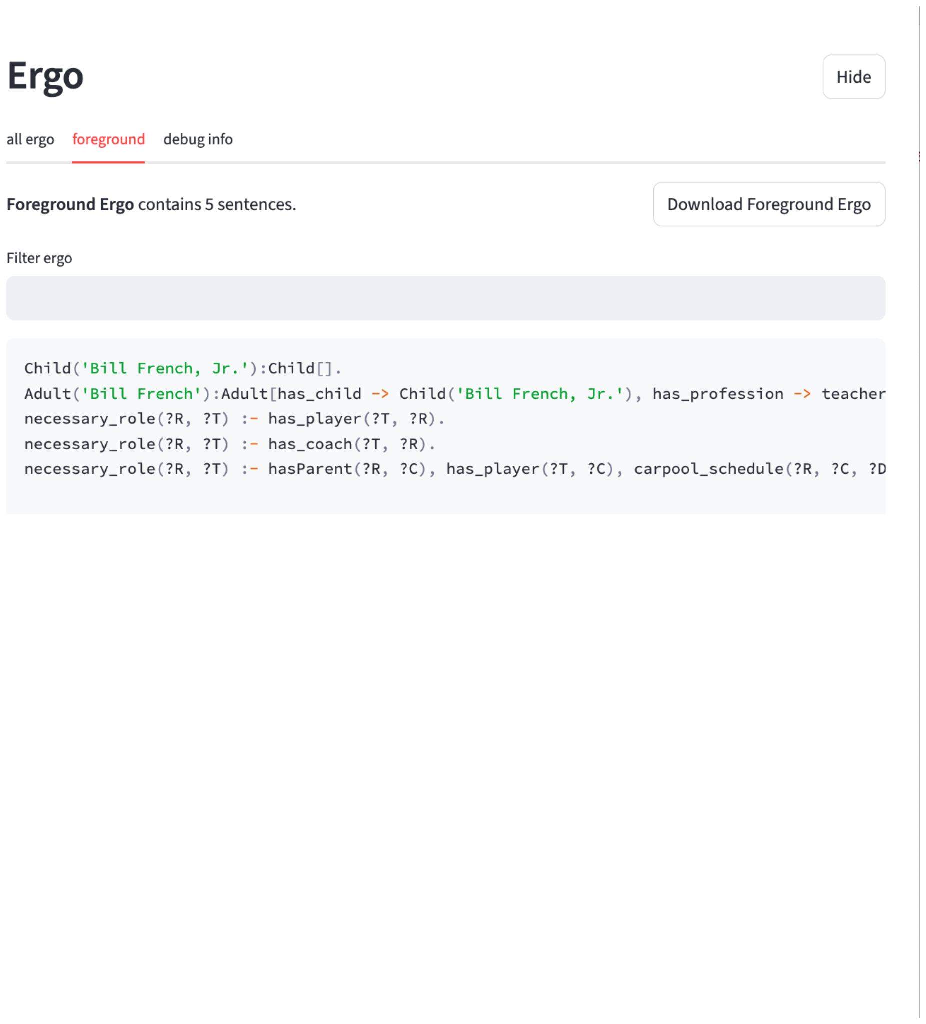
Rule in Ergo

```
Adult('Bill French'):Adult[has_pr                 > teacher, name -> 'Bill French'].
```

Natural Language description of a rule

Bill Jr. is the son of Bill French

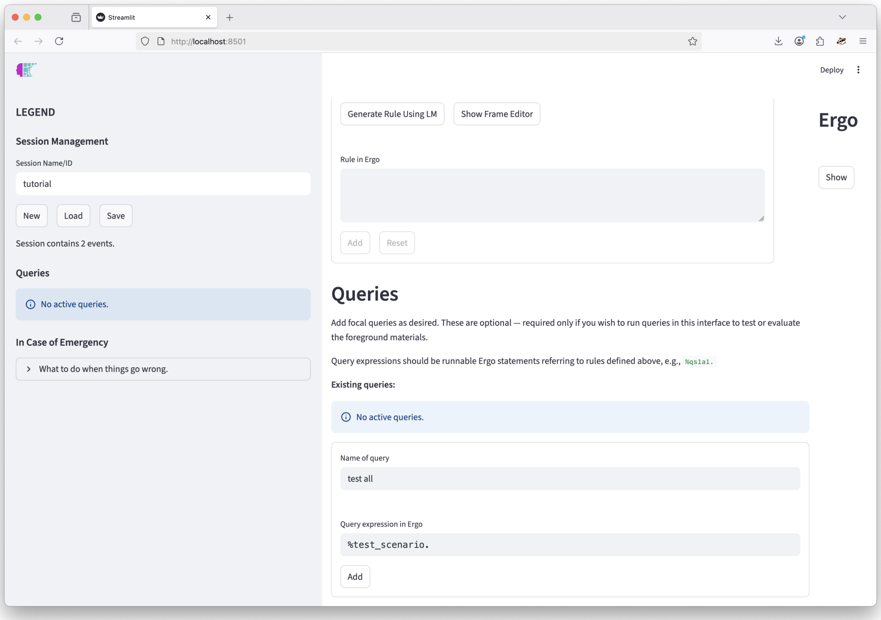
Figure 8: Adding Bill Jr. as Bill's son.



Foreground Ergo contains 5 sentences.

```
Child('Bill French, Jr.'):Child[].
Adult('Bill French'):Adult[has_child -> Child('Bill French, Jr.'), has_profession -> teacher
necessary_role(?R, ?T) :- has_player(?T, ?R).
necessary_role(?R, ?T) :- has_coach(?T, ?R).
necessary_role(?R, ?T) :- hasParent(?R, ?C), has_player(?T, ?C), carpool_schedule(?R, ?C, ?D
```

Figure 9: Accumulated foreground knowledge, showing rules and frame instances.



LEGEND

Session Management

Session Name/ID

tutorial

Session contains 2 events.

Queries

No active queries.

In Case of Emergency

What to do when things go wrong.

Rule in Ergo

Queries

Add focal queries as desired. These are optional — required only if you wish to run queries in this interface to test or evaluate the foreground materials.

Query expressions should be runnable Ergo statements referring to rules defined above, e.g., `%qs1a1.`

Existing queries:

No active queries.

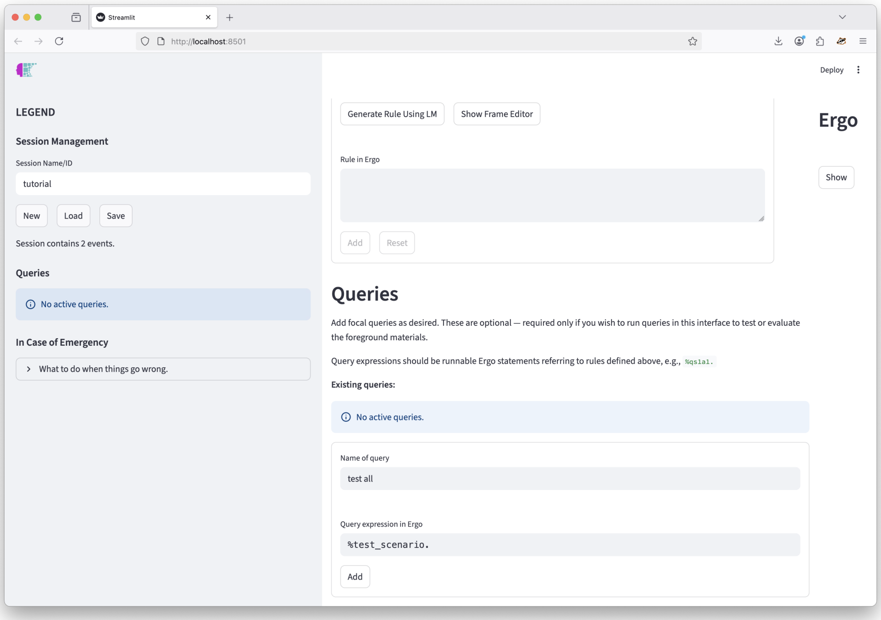
Name of query

test all

Query expression in Ergo

```
%test_scenario.
```

Figure 10: Adding a query from the soccer team warmup exercise.



LEGEND

Session Management

Session Name/ID

tutorial

Session contains 2 events.

Queries

No active queries.

In Case of Emergency

What to do when things go wrong.

Rule in Ergo

Queries

Add focal queries as desired. These are optional — required only if you wish to run queries in this interface to test or evaluate the foreground materials.

Query expressions should be runnable Ergo statements referring to rules defined above, e.g., `%qs1a1.`

Existing queries:

No active queries.

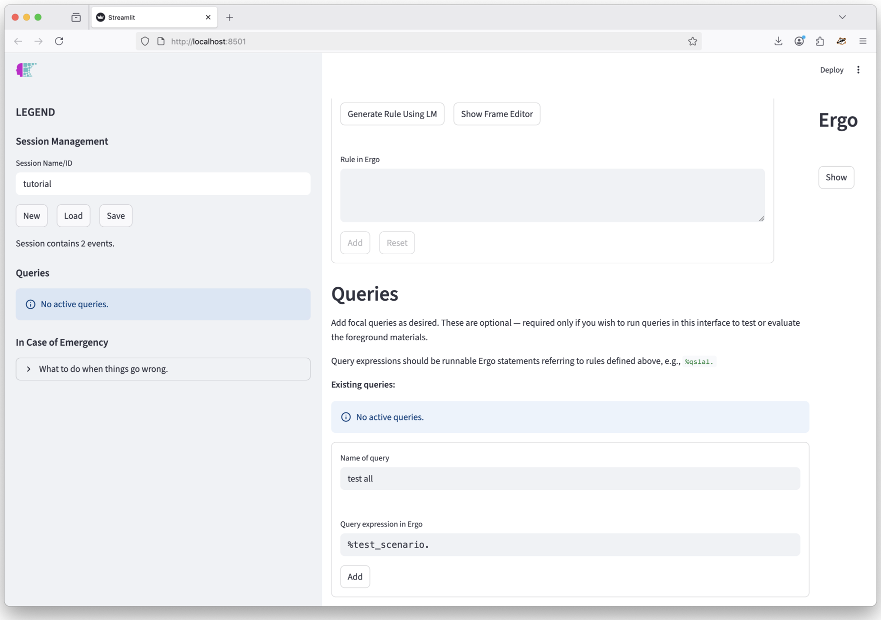
Name of query

test all

Query expression in Ergo

%test_scenario.

Figure 11: Adding a query from the soccer team warmup exercise.



Session Name/ID: tutorial

Session contains 2 events.

No active queries.

Rule in Ergo

Queries

Add focal queries as desired. These are optional — required only if you wish to run queries in this interface to test or evaluate the foreground materials.

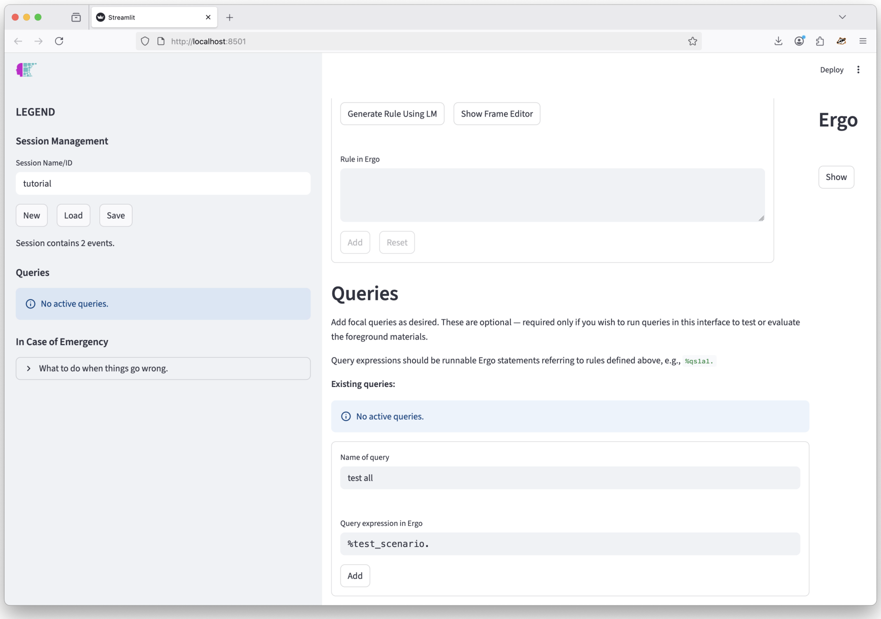
Query expressions should be runnable Ergo statements referring to rules defined above, e.g., `%qs1a1.`

Existing queries:

No active queries.

Name of query: test all

Query expression in Ergo: `%test_scenario.`



Session Name/ID: tutorial

Session contains 2 events.

No active queries.

Rule in Ergo

Queries

Add focal queries as desired. These are optional — required only if you wish to run queries in this interface to test or evaluate the foreground materials.

Query expressions should be runnable Ergo statements referring to rules defined above, e.g., `%qs1a1.`

Existing queries:

No active queries.

Name of query: test all

Query expression in Ergo: `%test_scenario.`

Figure 12: Browsing and downloading Ergo code.