CatLite Manual

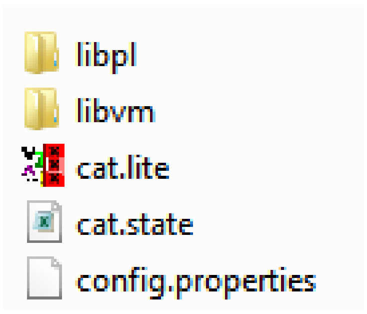
January 9, 2015

MDELite is a proposal to make tools for developing MDE applications – at least in a classroom setting – easier to use and less heavy-weight than Eclipse. Catalina is its 2nd-generation incarnation. MDELite was a code generator; Catalina is an interpreter. The difference in infrastructure is a growth from MDELite's miniscule core ~500 LOC to Catalina's ~5600 lines. Of course, there is quite a difference in appearance, but whether these extra capabilities were worth the effort remain to be seen.

CatLite is a subset of Catalina – although you have all the code for Catalina in a CatLite distribution, you only need the next section.

1. Catalina Application

To begin, you will be given a Catalina MDE application. It is a directory with the following contents:



- **libpl** is a directory containing some prolog files. There will be 2 files for each prolog database. A schema.pl file defines the tables of a prolog database – do not edit this file. The other file is a set of Prolog constraints that you will have to write. A dummy file which reports "you need to write real constraints" is given to you. You need to edit it.
- **libvm** is a directory containing velocity templates. You need to write real Velocity templates for each given file. A dummy file that reports you need write this file is given to you.
- **cat.lite.pl** contains a prolog database that Catalina interprets. Don't edit this file.
- **cat.state** is the violet file from which the cat.lite.pl file was generated. If you look at this file, you will see the arrows (transformations) that you can invoke.
- **config.properties** is needed by Catalina to get some basic coordinates before executing. Don't edit this file.

Bottom line: The only files that you are to edit are **libpl/*.conform.pl** and **libvm/*.vm**.

2. Command Line Operations of CatLite

You will be given a Prolog database as a starting point. Suppose this file is test1.rdb.pl. The name of the database is "test1". It conforms to the schema libpl/rdb.schema.pl, which you can find in your application directory.

There are very few command-line invocations that you will use. Here they are:

Conformance. To test if a database satisfies constraints, run:

```
> java CatCore.Main test1.rdb.pl conform
```

If there are conformance errors, they'll be printed out. A silent execution means that the prolog database satisfies its database constraints.

Velocity. To invoke Velocity, there are two ways:

```
> java CatCore.Main test1.rdb.pl run toCode
```

Velocity is the CatLite tool that inputs a prolog database and executes a velocity template to produce output. The above line says: test1.rdb.pl is the prolog database and the velocity template that is to be executed is libvm/toCode.vm. The CatLite tools are designed only to execute arrows (transformations) that are predefined. The only arrows that CatLite will execute are the X values in libvm/X.vm. If you want to invoke other Velocity templates – say for reasons of testing or debugging, you need another way.

The vm2t tool needs schema information for it to build its internal tables. Again consider the test1.rdb.pl file. The rdb schema is (in the Catalina universe) is libpl/rdb.schema.pl. So what you do is run this script to merge the rdb schema definition with your prolog database, and then feed that to vm2t tool directly:

```
> cat libpl/rdb.schema.pl test1.rdb.pl > my.pl

> java CatCore.vm2t.Main my.pl X.vm bbb
```

Where "X.vm" is the relative path to your velocity template and "bbb" is the name of your output file.

Filth. Catalina generates all sorts of files during its execution. Most are for debugging, but not all. Never the less, Catalina-generated files clog up a directory. To clean out its files, run:

```
> java CatCore.Clean
```

That's it. Good luck!