Beginning to use the clockwork programming language

Martin Leadbeater, Sep 2013

This tutorial aims to show how clockwork and its related tools can be used to explore some sample programs.

Table of Contents

1 Getting Started

The clockwork programming language is intended for monitoring and controlling systems and as such, clockwork programs normally run continuously. The distribution includes a small program: "hello.cw" that displays "Hello World" and then exits. The program script is:

```
Hello MACHINE { ENTER INIT { LOG "Hello World"; SHUTDOWN } }
hello Hello;
```

This following subsections aim to help you execute the hello.cw program.

1.1 Install clockwork

- git clone http://github.com/latproc/clockwork
- cd clockwork/iod
- make -f Makefile.cw

1.2 Run clockwork

- cd ../tests
- ../iod/cw hello.cw

The program will display a hello message:

```
2013-09-03 15:39:36.689 ------- hello: Hello World -------
```

2 Using the interactive shell

Clockwork programs run continuously and it is possible to interact with them using a number of techniques:

- the interactive shell (iosh)
- the web user interface (to be covered later in this document)
- using ØMQ (zeromq)
- using modbus

The simplest of these to start with is iosh.

2.1 Starting iosh

- execute the steps in section 1.1, above; those step build 'cw' and 'iosh'
- run iod/iosh (if you are still in the clockwork/iod directory, simply type

  ./iosh

You should see a prompt like the following:

```
Connecting to tcp://127.0.0.1:5555

Enter HELP; for help. Note that ';' is required at the end of each command
  use exit; or ctrl-D to exit this program
```

1. enter HELP; to display a list of available commands
2. in another terminal, run clockwork against one of the test files in the clockwork/tests directory:

   ```
   ./cw ../tests/blink.cw
   ```

3. back in iosh, enter LIST; to display a list of machines that the clockwork program is running;

   ```
   > LIST;                 type this
   blinker Blinker         clockwork responds with this
   ```

4. Look at the source for the blink program:

```
1     Blinker MACHINE {
2       on STATE;
3       off STATE;
4       ENTER on { WAIT 1000; SET SELF TO off; }
5       ENTER off { WAIT 1000; SET SELF TO on; }
6     }
7     blinker Blinker;
```

Note that the machine has two states, 'on' and 'off' the two ENTER functions cause the machine to flip between on and off after a delay of 1000ms.

5. you can also describe the internals of the blinker machine:

```
1     > DESCRIBE blinker;
2     > ---------------
3     blinker: INIT
4       Class: Blinker instantiated at: blink.cw line:10
5     properties:
6       NAME:blinker,STATE:INIT
7
8     Timer: 24503
```

From this description you can see that the machine called 'blinker' is in a state 'INIT' (line 3), also, blinker has been in the INIT state for 24.503 seconds.

When clockwork starts running, all machines enter a state called INIT. Normally they quickly change state automatically but since the 'blinker' has nothing that causes a state change to 'on' or 'off', it will stay in INIT forever.

6. manually set the blinker into the 'on' state using the SET command in iosh:

```
> SET blinker TO on;
```

7. you can also get the state of the blinker using the GET command:

```
> GET blinker;
> on
> GET blinker;
> off
```

Note: If the iosh command prompt '>' does not come back, iosh is trying to connect to an instance of clockwork but can't; check that you still have cw running on the local host.

3 Monitoring clockwork while it is running

Clockwork uses the ØMQ messaging library for most of its communication needs, for example, iosh communicates with clockwork via ØMQ. Whenever machines change state, clockwork sends a message on a ØMQ channel that other programs can subscribe to. The distribution includes a sample program that monitors this channel and displays state change information on the terminal.

This example runs from the top-level of the clockwork distribution.

1. execute the steps in section 1.1, above; those step build 'cw' and 'iosh'
2. in another terminal, run clockwork against the blink.cw files in the clockwork/tests directory:

```
iod/cw ../tests/blink.cw
```

3. use iosh to start the blinker working:

```
echo 'SET blinker TO on;' | iod/iosh
```

4. run the message monitoring program:

```
$ iod/zmq_ecat_monitor
Listening on port 5556
```

```
blinker  STATE on
blinker  STATE off
blinker  STATE on
blinker  STATE off
```

The monitor can be very useful (and very verbose) when testing clockwork programs but we have also recently started a new project at http://github.com/latproc/scope that aims to make much more sophisticated monitoring tools.

4 Interacting with clockwork using a web client

The clockwork distribution includes a copy of mini_httpd (http://acme.com/software/mini_httpd/) that we have configured to provide a server for web-based interaction. There are a few other things to do to setup the server however and you may be more comfortable using another system. The critical components are: php 5.3 and the zmq library for php (http://zeromq.org/bindings:php).

[TBD more information required]

The web site organizes groups of clockwork machines into tabs and also provides some other tabs relating the Beckhoff EtherCAT™ system if it is being used. To keep the web page in sync with clockwork, the client periodically issues a request for a list of all machines, their states and properties.

For example, here is the LIST JSON output for our blink.cw program

```
> LIST JSON;
> [{
            "name":     "blinker",
            "class":    "Blinker",
            "NAME":     "blinker",
            "STATE":    "on",
            "executing":      "WaitAction 1000",
            "enabled":  true,
            "state":    "on"
      }]
```