TuCSoN4Jason quick guide

In this quick guide, you will learn how to get TuCSoN4Jason (**t4jn** for short), build it and run a first example showcasing its features. You will also learn the main API available to Jason developers and how to use them in your code, for your Jason agents, as well as a bit of "behind the scenes" on how t4jn works.

Assumptions are you are familiar with Java (compilation), Jason (usage), Git (cloning repositories), optionally, ANT (buildfiles), Jason and TuCSoN.

---

1. Getting Started with t4jn

1.1 Downloading

If you want the *ready-to-use* distribution of t4jn, download **t4jn.jar** archive from the "Downloads" section, here > http://bitbucket.org/smariani/tucson4jason/downloads.

If you want the *source code* of t4jn, clone the t4jn **Git repository** hosted here on Bitbucket, at > http://smariani@bitbucket.org/smariani/tucson4jason.git (e.g., from a command prompt type `$> git clone https://smariani@bitbucket.org/smariani/tucson4jason.git`) [1].

In the former case, skip to the "Running" section below. In the latter case, keep reading.

1.2 Compiling

By cloning t4jn you have downloaded a folder named `tucson4jason/`, with the following *directory structure*:

```
tucson4jason/
|__...
|__jason-tucson/
   |__...
   |__ant-scripts/
      |__build.xml
      |__environment.properties
   |__eclipse-config/
   |__how-to/
   |__license-info/
   |__src/
```

t4jn depends on 3 other Java libraries to function properly [2]:

- **Jason**, downloadable from Jason "Download" page, here > http://sourceforge.net/projects/jason/files/ (**jason.jar**)
- **TuCSoN**, downloadable from TuCSoN "Downloads" section, here > http://bitbucket.org/smariani/tucson/downloads (**tucson.jar**)
- **tuProlog**, downloadable from tuProlog "Download" page, here > http://apice.unibo.it/xwiki/bin/view/Tuprolog/Download (**2p.jar**)

Once you got the above libraries, you are ready to compile t4jn source code.

The easiest way to do so is by exploiting the ANT script named `build.xml` within folder `ant-scripts/`, which takes care of the whole building process for you, from compilation to deployment (covered in next section). To do so, you need to have ANT installed on your machine [3]. If you don't want to use ANT, build t4jn jar archive using the tools you prefer, then skip to the "Running" section below.

To compile t4jn using ANT:

1. Edit the `environment.properties` file according to your system configuration:

   1.1 Tell ANT where your JDK and your `java` tool are

   1.2 Tell ANT which libraries are needed to compile t4jn (those you just downloaded, that is Jason, TuCSoN and tuProlog)

   1.3 Tell ANT where you put such libraries (e.g. if you put them into `jason-tucson/libs/` you are already set)

   *[1.4 Tell ANT your Bitbucket username (for automatic syncing with t4jn repository, **not supported at the moment**)]*

2. Launch the ANT script using target `compile` (e.g., from a command prompt position yourself into the `ant-scripts/` folder then type `$> ant compile`) [4]. This will create folder `classes/` within folder `jason-tucson/` and therein store Java `.class` files.

Other ANT targets are available through the `build.xml` file: to learn which, launch the ANT script using target `help`.

1.3 Deploying

Deploying t4jn is as simple as giving a different build target to the ANT script `build.xml`:

- if you only want the **t4jn jar** archive, ready to be included in your Jason project, launch the script using target `lib`. This will compile t4jn source code into binaries (put into `jason-tucson/classes/` folder) then package them to **t4jn.jar** into `jason-tucson/lib/` folder.
- if you want a **ready-to-release distribution** of t4jn, including also documentation and support libraries, launch the script using target `dist`. This will:
    - compile t4jn source code into binaries, put into `jason-tucson/classes/` folder
    - package them to t4jn.jar, put into `jason-tucson/lib/` folder
    - generate Javadoc information, put into `jason-tucson/doc/` folder
    - create folder `rel/TuCSoN4Jason-${version}` including:
        - folder `docs/` including the generated Javadoc information as well as this "how-to"
        - folder `libs/` including Jason, TuCSoN and tuProlog libraries used to build t4jn
        - folder `rel/` including t4jn jar archives and example programs

The complete directory structure obtained by launching `ant dist` build process should look like the following (assuming you put Jason, TuCSoN and tuProlog libraries in folder `jason-tucson/libs/`):

```
tucson4jason/
|__...
|__jason-tucson/
   |__...
   |__ant-scripts/
      |__build.xml
      |__environment.properties
   |__classes/
   |__doc/
   |__eclipse-config/
   |__how-to/
   |__rel/
      |__TuCSoN4Jason-${version}/
         |__docs/
            |__how-to/
            |__javadoc/
         |__libs/
         |__rel/
      |__...
   |__lib/
   |__libs/
   |__license-info/
   |__src/
```

Other ANT targets are available through the `build.xml` file: to learn which, launch the ANT script using target `help`.

1.4 Running

To run t4jn, you need:

- t4jn jar, **t4jn.jar**
- Jason jar, **jason.jar**
- TuCSoN jar, **tucson.jar**
- tuProlog jar, **2p.jar**

Supposing you built t4jn using the provided ANT script [5] and that you are comfortable with using a command prompt to launch Java applications [6]:

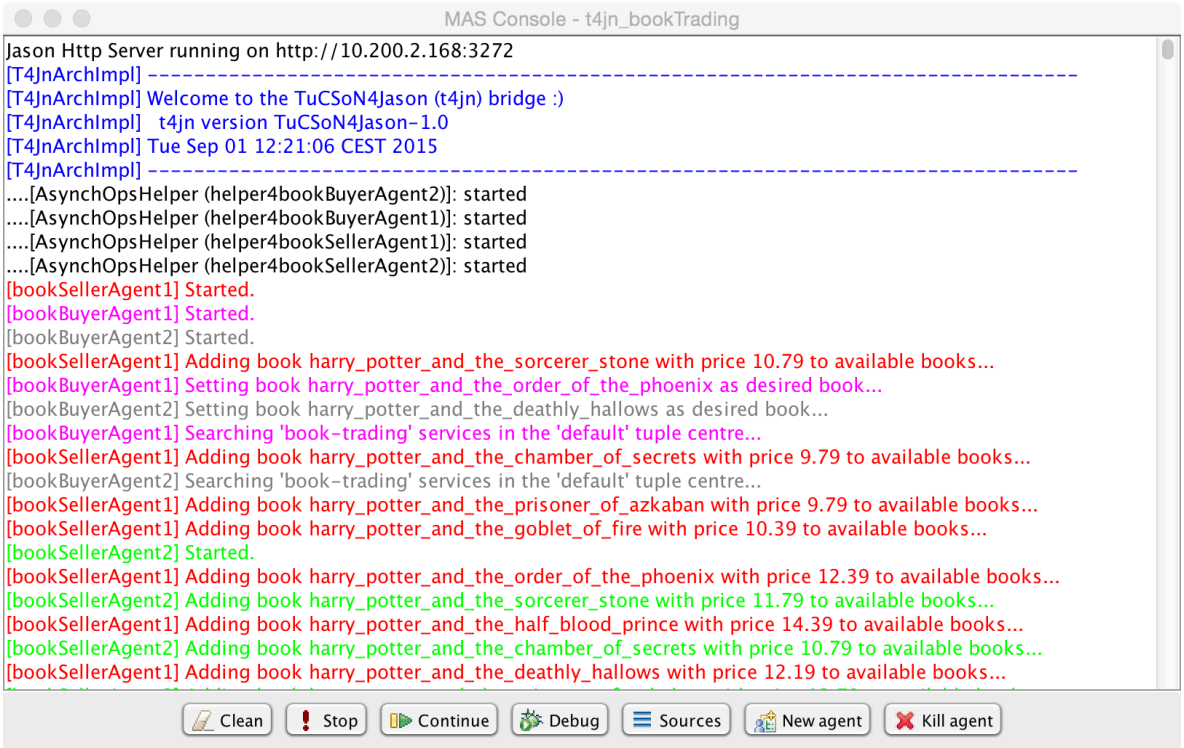
1. open a command prompt and position yourself into `jason-tucson/`
2. launch a TuCSoN node on the default address, as follows [7]:

```
java -cp libs/tucson.jar:libs/2p.jar alice.tucson.service.TucsonNodeService
```

3. open a new tab/window of the command prompt and position yourself into either `jason-tucson/lib/` or `jason-tucson/rel/TuCSoN4Jason-${version}/rel/` folder
4. launch the example application "Book Trading" MAS (MAS configuration file `t4jn_bookTrading.mas2j`), shipped within t4jn.jar, showcasing its features, as follows [7]:

```
java -cp t4jn.jar:../libs/jason.jar:../libs/2p.jar:../libs/tucson.jar
jason.infra.centralised.RunCentralisedMAS t4jn_bookTrading.mas2j
```

Jason GUI should appear with the t4jn welcome banner, as depicted below.



You should see many prints on Jason GUI, tracking what happens in the MAS.

---

2. Using t4jn

1.1 API Overview

The first step in integrating TuCSoN and Jason has been implementing a **custom Jason agent architecture**: this is needed for the integration of TuCSoN *interaction model* (suspensive primitives) with Jason *concurrency model* (intentions interleaving) to be *autonomy-preserving* -- more on this in [8].

This means TuCSoN4Jason `T4JnArchImpl` class (hidden to clients), representing the bridge between Jason's transition system and TuCSoN coordination services, extends Jason `AgArch` class, the base class representing Jason agents' BDI reasoning architecture. As a bridge, this class handles all the communication and coordination between the two technologies: Jason agents requests for TuCSoN coordination services, their replies, whether to suspend the caller intention and when to resume it.

The second step has been implementing **custom internal actions**, composing the public API Jason agents would use to interact with TuCSoN. All put in package `t4jn.api` (to shorten code for invocation from Jason), they are the following (in Prolog-like API notation) -- more to come, to complete the corresponding TuCSoN primitives API:

- `out/4`, invoking TuCSoN `out` primitive. Arguments configuration by index is: 0 name of the tuple centre target of the operation (as a Jason string), 1 > IP address of the TuCSoN node hosting it (as a Jason string), 2 > TCP port number where the TuCSoN node is active (as a Jason string), 3 > the tuple argument of the operation (as a Jason `entry`), 4 > a variable to be unified the unique (per-agent) ID of the invocation -- for Jason agents to get back the result.
- `in/4`, invoking TuCSoN `in` primitive. Arguments as above.
- `rd/4`, invoking TuCSoN `rd` primitive. Arguments as above.
- `no/4`, invoking TuCSoN `no` primitive. Arguments as above.
- `inp/4`, invoking TuCSoN `inp` primitive. Arguments as above.
- `rdp/4`, invoking TuCSoN `rdp` primitive. Arguments as above.
- `nop/4`, invoking TuCSoN `nop` primitive. Arguments as above.
- `outAll/4`, invoking TuCSoN `outAll` primitive. Arguments as above.
- `inAll/4`, invoking TuCSoN `inAll` primitive. Arguments as above.
- `rdAll/4`, invoking TuCSoN `rdAll` primitive. Arguments as above.
- `spawn/4`, invoking TuCSoN `spawn` primitive. Arguments as above.
- `set/4`, invoking TuCSoN `set` primitive. Arguments as above.
- `get/3`, invoking TuCSoN `get` primitive. Arguments configuration by index is: 0 name of the tuple centre target of the operation, 1 > IP address of the TuCSoN node hosting it, 2 > TCP port number where the TuCSoN node is active, 3 > a variable to be unified the unique (per-agent) ID of the invocation -- the tuple argument isn't needed due to `get` semantics, see TuCSoN API documentation.
- `getArg/3`, to retrieve the argument of a tuple at the given index. Arguments configuration by index is: 0 > the tuple to inspect, 1 > the index of its desired argument, 2 > a variable to be unified with the requested argument. Notice TuCSoN logic tuples are perfectly interoperable with Jason `entry` data type, thus, this internal action as well as all the others here listed can be used with Jason entries as well.
- `getResult/2`, to retrieve the result of a previously invoked TuCSoN operation. Arguments configuration by index is: 0 > TuCSoN invocation ID, 1 > a variable to be unified with the

result.

The reason why the latter operation is needed, and why operation results are NOT a return parameter of the corresponding operation itself, is because *TuCSoN operations are suspensive*, thus can block the caller Jason agent -- **the WHOLE agent**, not only the caller intention! Therefore, "under the hood", we had to: (i) intercept TuCSoN primitives calls -- done by our custom agent architecture `T4JnArchImpl`; (ii) **make them asynchronous** -- exploiting TuCSoN asynchronous support, see [9]); (iii) keep track of the results asynchronously received by our bridge -- again, done by `T4JnArchImpl`; (iv) provide some means to get such results when needed -- indeed, primitive `getResult`.

Be aware that such primitive is the only one with a **suspensive semantics**: if the result of the operation queried with `getResult` is NOT available yet, the **caller intention gets suspended**, to be resumed as soon as the operation result becomes available -- which is automatically unified with `getResult` return variable to be used in the remainder of the plan.

1.1 "Hands-on" step-by-step tour

NB: What follows is based on the "Book Trading" example scenario shipped within t4jn distribution (see t4jn "Getting Started", here).

[TBD]

Contact Information

Author of this "how-to":

- *Stefano Mariani*, DISI - Università di Bologna (s.mariani@unibo.it)

Authors of the add-on:

- *Stefano Mariani*, DISI - Università di Bologna (s.mariani@unibo.it)
- Federico Foschini, Università di Bologna
- Alessandro Fantini, Università di Bologna
- Prof. Andrea Omicini, DISI - Università di Bologna

---

[1] Git standalone clients are available for any platform (e.g., SourceTree for Mac OS and Windows). Also, if you are using Eclipse IDE for developing in Jason, the EGit plugin is included in the Java Developers version of the IDE.

[2] Recommended Jason version is **1.4.2**. Regarding TuCSoN and tuProlog, recommended version is **1.12.0.0301** and **3.0.1**, respectively. Others (both newer and older) may work properly, but they have not been tested.

[3] Binaries available here, installation instructions covering Linux, MacOS X, Windows and Unix systems here.

[4] If you are using Eclipse IDE for developing in Jason, ANT is included: click "Window > Show View > Ant" then click "Add buildfiles" from the ANT view and select file `build.xml` within `ant-scripts/` folder. Now expand the "TuCSoN4Jason build file" from the ANT view and finally double click on target `compile` to start the build process.

[5] If you directly downloaded t4jn jar or if you built it from sources without using the provided ANT script, simply adjust the given command to suit your configuration.

[6] If you do not want to use the command prompt to launch Jason applications, adjust the given command to suit your configuration, e.g., if your are using Eclipse IDE: right-click on "tucson.jar > Run As > Run Configurations..." then double-click on "Java Application", select "TucsonNodeService - alice.tucson.service" as the main class (libs/tucson.jar:libs/2p.jar is automatically added by Eclipse according to project's build path settings); then, right click on "t4jn_bookTrading.mas2j > Run Jason Application".

[7] Separator `:` works on Mac & Linux only, use `;` on Windows.

[8] Mariani, S., Omicini, A., Sangiorgi, L.: "*Models of Autonomy and Coordination: Integrating Subjective & Objective Approaches in Agent Development Frameworks*". Published in 8th International Symposium on Intelligent Distributed Computing (IDC 2014), 3-5 September 2014. Available from here.

[9] http://apice.unibo.it/xwiki/bin/download/TuCSoN/Documents/asynchronous%2Dsupport.pdf

---