

## How to use

### CLIF v2 and ISAC plug-ins



with the Eclipse™ console

[clif.ow2.org](http://clif.ow2.org)

Copyright © 2006-2011 France Telecom

License information: <http://creativecommons.org/licenses/by-nc-sa/3.0>

## Table of contents

<b>1. Introduction to ISAC Plug-ins.....</b>	<b>3</b>
1.1. What is an ISAC Plug-ins.....	3
1.2. Installation.....	3
<b>2. ISAC is a Scenario Architecture for CLIF.....</b>	<b>4</b>
2.1. Defining an ISAC scenario for LDAP.....	5
2.2. Define your ISAC scenario.....	6
2.2.1. <i>Create your test project and your ISAC scenario.....</i>	6
2.2.2. <i>Import ISAC plug-ins.....</i>	9
2.2.3. <i>Define your behavior.....</i>	13
2.2.4. <i>Load Profiles.....</i>	21
2.3. Define the test plan.....	24
2.3.1. <i>Rationale.....</i>	24
2.3.2. <i>Create a test plan.....</i>	24
2.3.3. <i>Add Probes and Injectors to the test plan.....</i>	25
2.4. Deploying and executing the test plan.....	28
<b>3. Browsing test results.....</b>	<b>32</b>
3.1. Getting raw measures.....	32
3.2. Getting monitoring data.....	32
<b>4. CLIF servers and registry.....</b>	<b>33</b>
4.1. Installation.....	33
4.2. Rationale.....	33
4.3. Running a registry.....	33
4.4. Configuring a CLIF server.....	33
4.5. Running a CLIF server.....	33

# 1. Introduction to ISAC Plug-ins

## 1.1. What is an ISAC Plug-ins

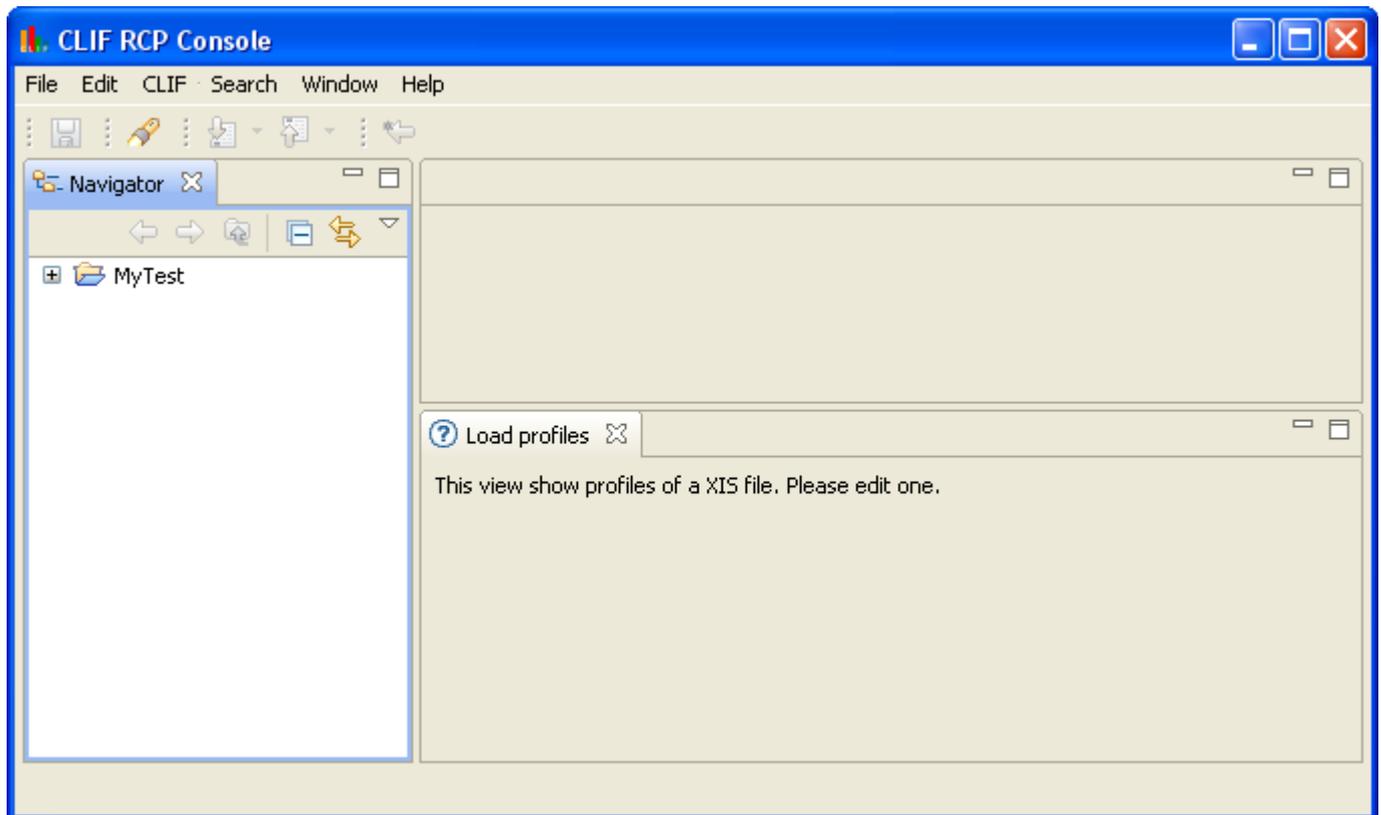
In order to actually generate traffic on a System Under Test (SUT), we need to define a behavior. A behavior can be understood as a logical definition, a kind of a skeleton. This skeleton must be associated to one or more ISAC plug-ins. Plug-ins are external Java libraries, that are responsible for:

- performing actions (i.e. generating requests) on the SUT, using and managing specific protocols whose response times will be measured (e.g. HTTP, DNS, JDBC, TCP/IP, SIP, LDAP);
- providing conditions used by the behaviors' conditional statements (if-then-else, while, preemptive);
- providing timers to implement delays (think time), for example with specific random distributions or computed in some arbitrary way;
- providing ad hoc controls for the plug-in itself (e.g. to change some settings);
- providing support for external data provisioning (e.g. a database of product references or a file containing identifier-password pairs for some user accounts), used as parameters by the behaviors.

## 1.2. Installation

See Installation Manual for Eclipse-based console GUI installation.

## 2. ISAC is a Scenario Architecture for CLIF

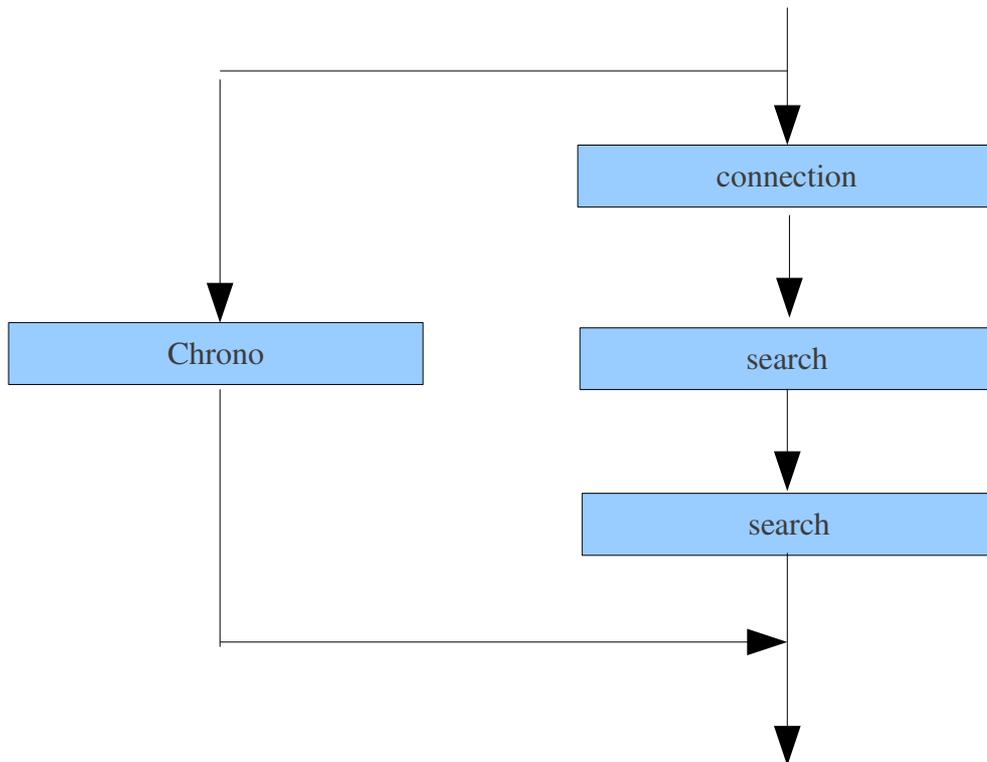


With ISAC, testers are given a way to define load scenarios by combining:

- definitions of elementary behaviors, typically representing users;
- optional definitions of load profiles setting the population (i.e. the number of active instances) of each behavior as a function of time.

## 2.1. Defining an ISAC scenario for LDAP

In this tutorial, we will define an ISAC scenario which will send some requests on the LDAP directory. We can see below the actions that our scenario will do:



In this scenario we will connect to the LDAP directory, make a search and disconnect it. We will import the `LdapInjector` plug-in to be able to do this action.

To externalize the data used in the scenario (mandatory parameters of the `LdapInjector` plug-in) we will create CSV file that will contain all the needed data. Each line of the CSV file represents the parameters needed to execute the scenario. Then we will loop on each line to make different calls to the LDAP directory. To do this we will use the `CSVProvider` plug-in.

In this ISAC scenario the scenario duration will be set to 1 second. We need to import the `ConstantTimer` plug-in.

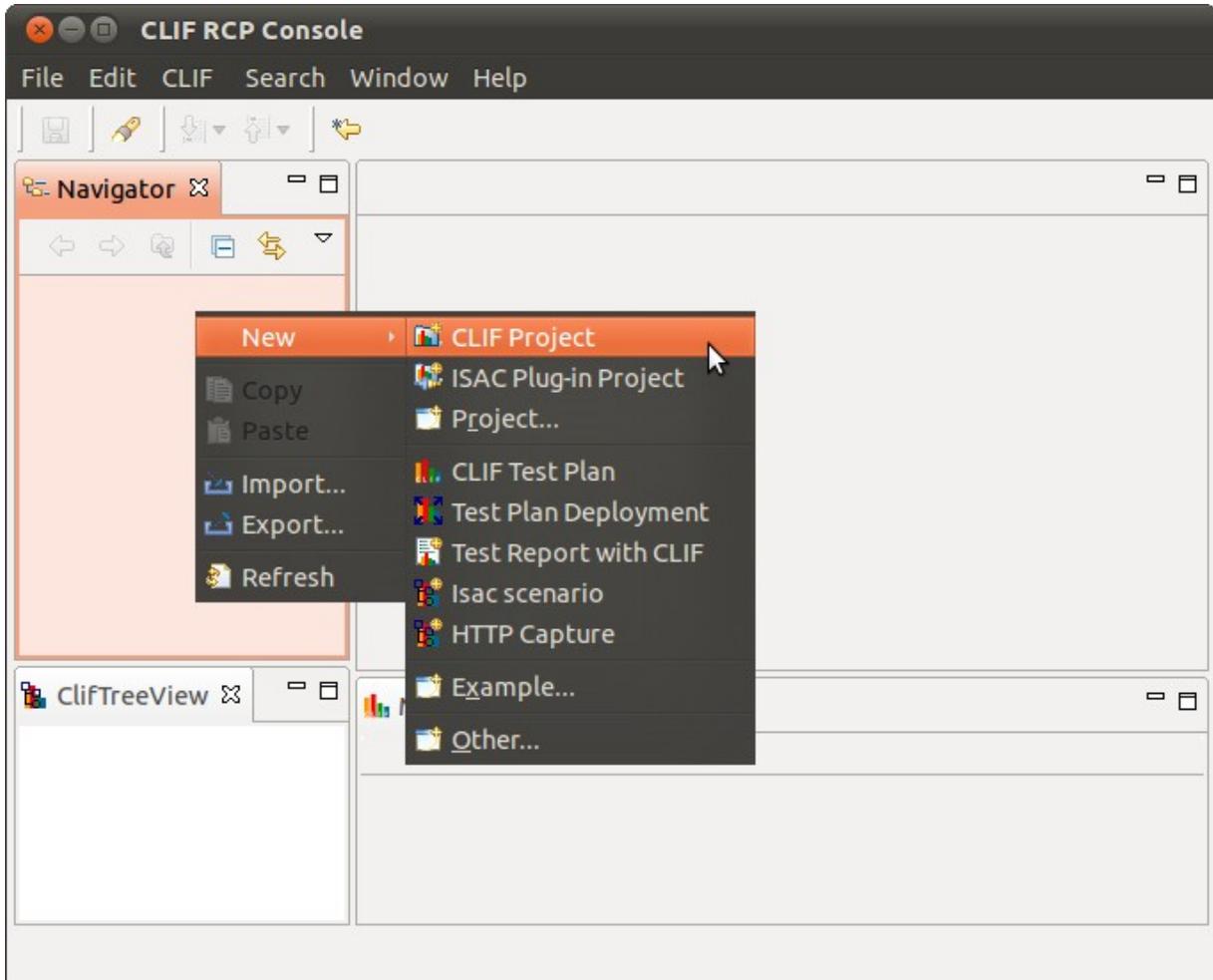
In our load test we will use two or more injectors and we don't want to launch the test at the same moment on each injector. So we will import the `Random` plug-in.

Finally, we want to know the response time of group of operations, that's why we will import the `Chrono` plug-in.

## 2.2. Define your ISAC scenario

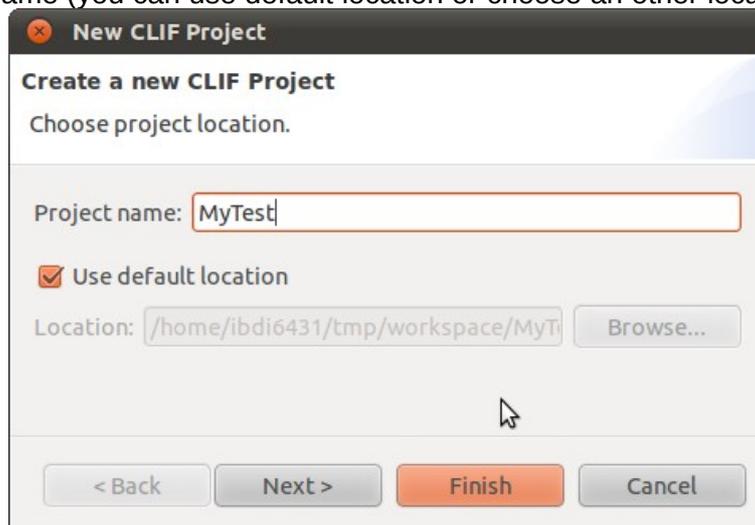
### 2.2.1. Create your test project and your ISAC scenario

First of all, you have to create a new CLIF project. Click on File -> New -> CLIF project



Enter your project name (you can use default location or choose an other location)

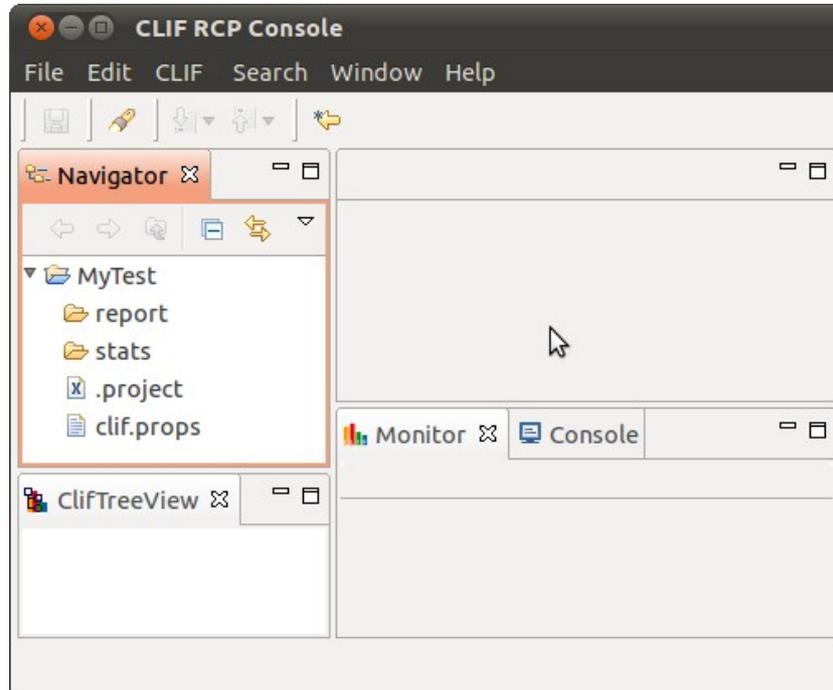
Click on Finish.



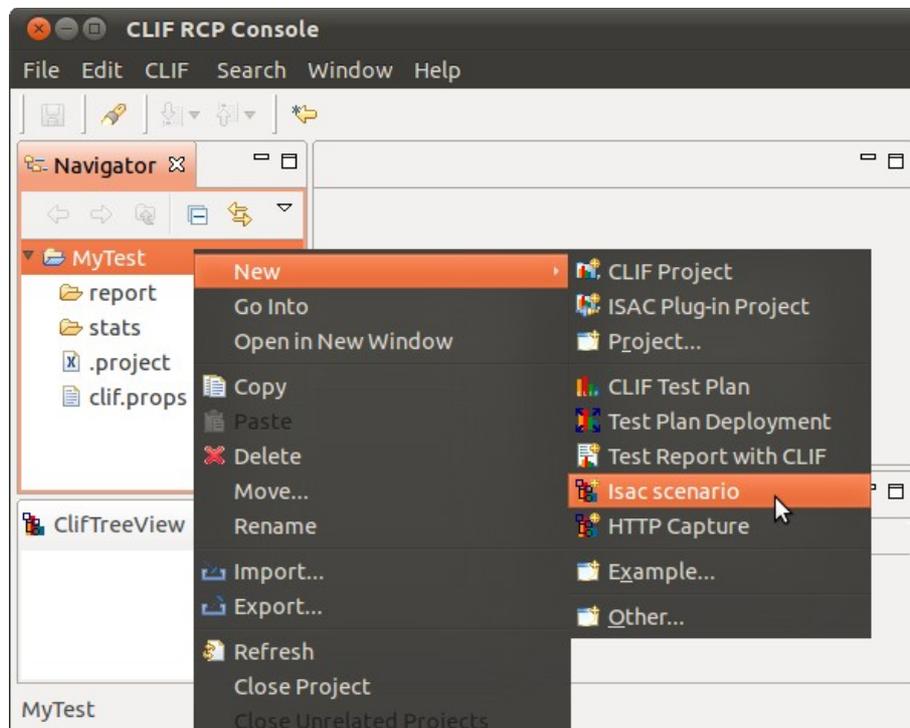
The console window is split into several panels, called views. You may have a variety of specific assemblies of views, called perspectives. You can call a specific view or perspective from the Window menu. CLIF comes with two perspectives:

- the CLIF perspective is dedicated to defining CLIF test plans and run test executions;
- the ISAC perspective is dedicated to the creation of ISAC scenarios.

You should be now in the CLIF perspective:

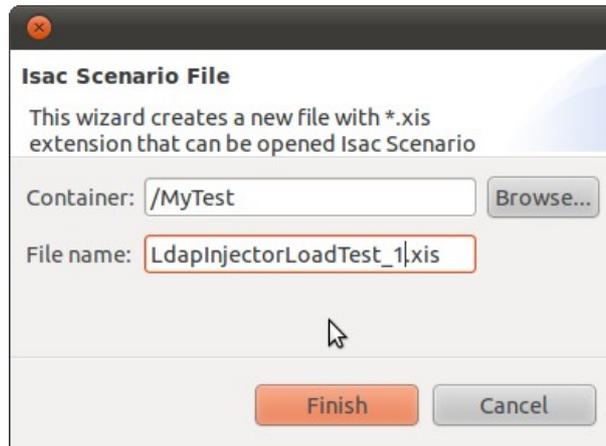


To create an Isac scenario, click on File -> New -> Isac Scenario:

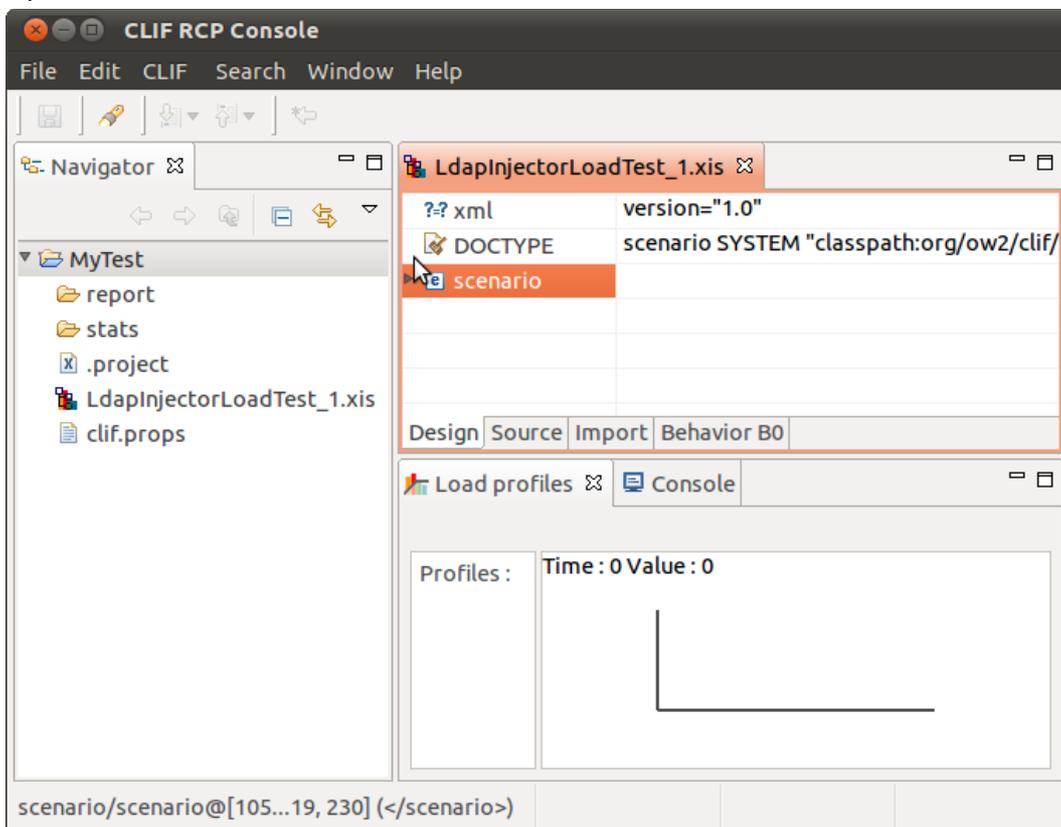


## How To Use CLIF v2 and ISAC plug-ins

Choose the container (basically the target project's directory) and give a name to your ISAC scenario.



Click on Finish. Your window layout should automatically switch to the set of views defined by the ISAC perspective:



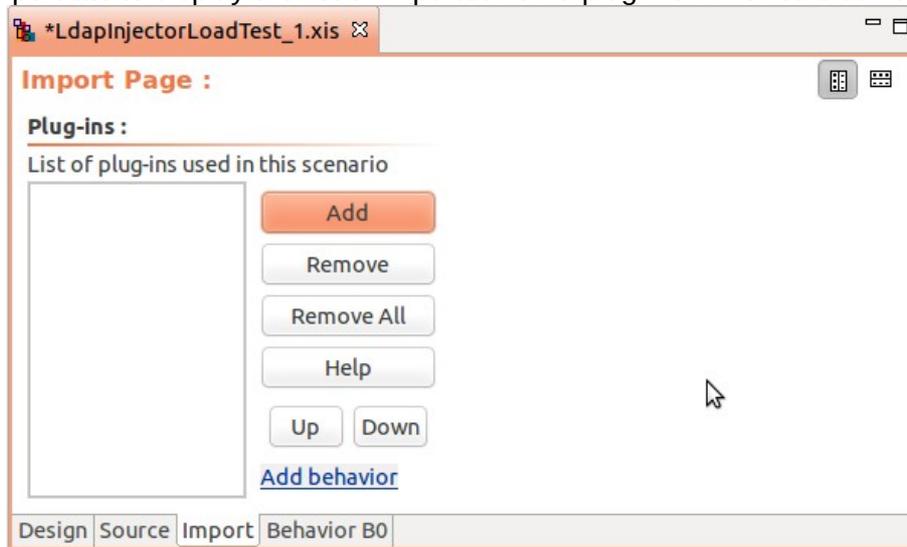
Now you can see the content of your scenario file in the ISAC view. At the bottom of this view, you can see four tabs:

- "Design" shows the tree structure of the xis file (which is an xml file),
- "Source" is an XML editor for the xis file,
- "Import" allows you to import the ISAC plug-ins that you will need in your scenario,
- "Behavior B0" is an editor for the default "B0" behavior. It defines your virtual users behavior and associated load profile. You may define more than one behavior, and give arbitrary names to behaviors. Each behavior is edited in a separate tab.

Below the ISAC view, you see the Load profile view, with currently no load profile defined.

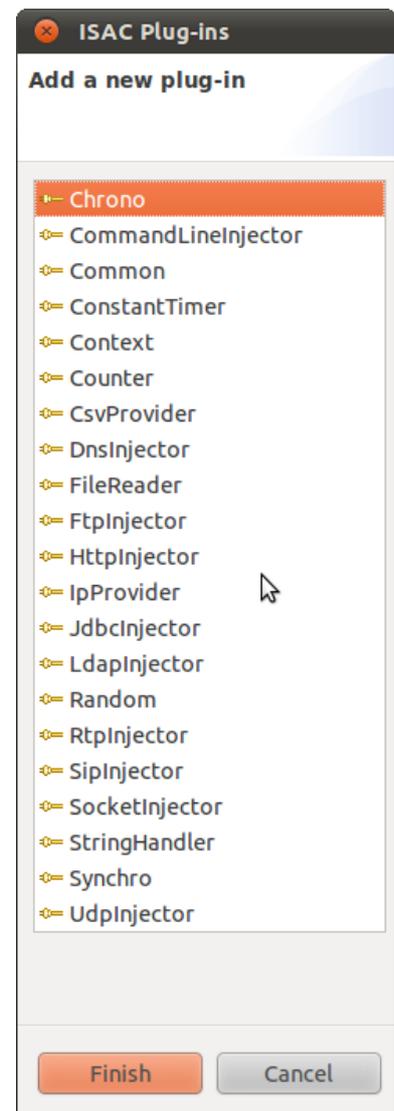
## 2.2.2. Import ISAC plug-ins

Click on the Import tab to display the list of imported ISAC plug-ins. This list is currently empty.



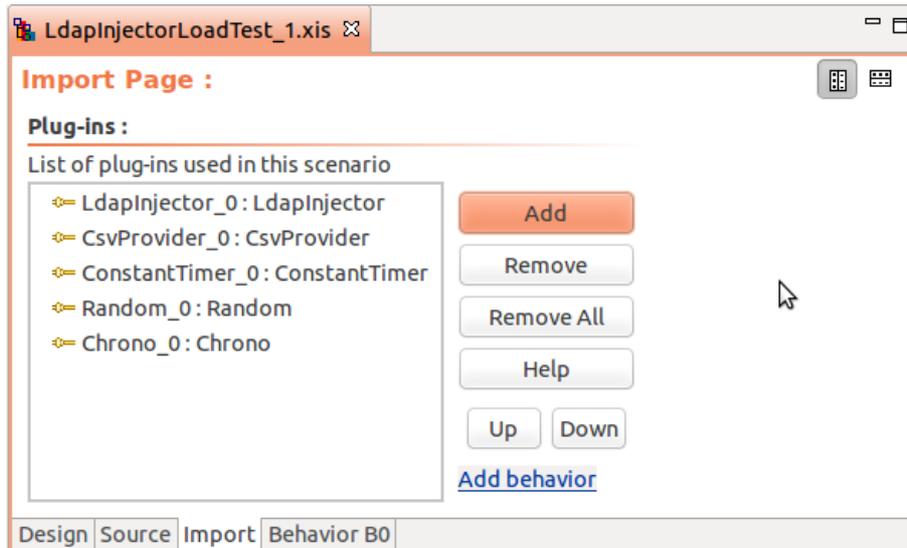
Click on the Add button. Import the following plug-ins one by one:

- LdapInjector
- CSVProvider
- ConstantTimer
- Random
- Chrono

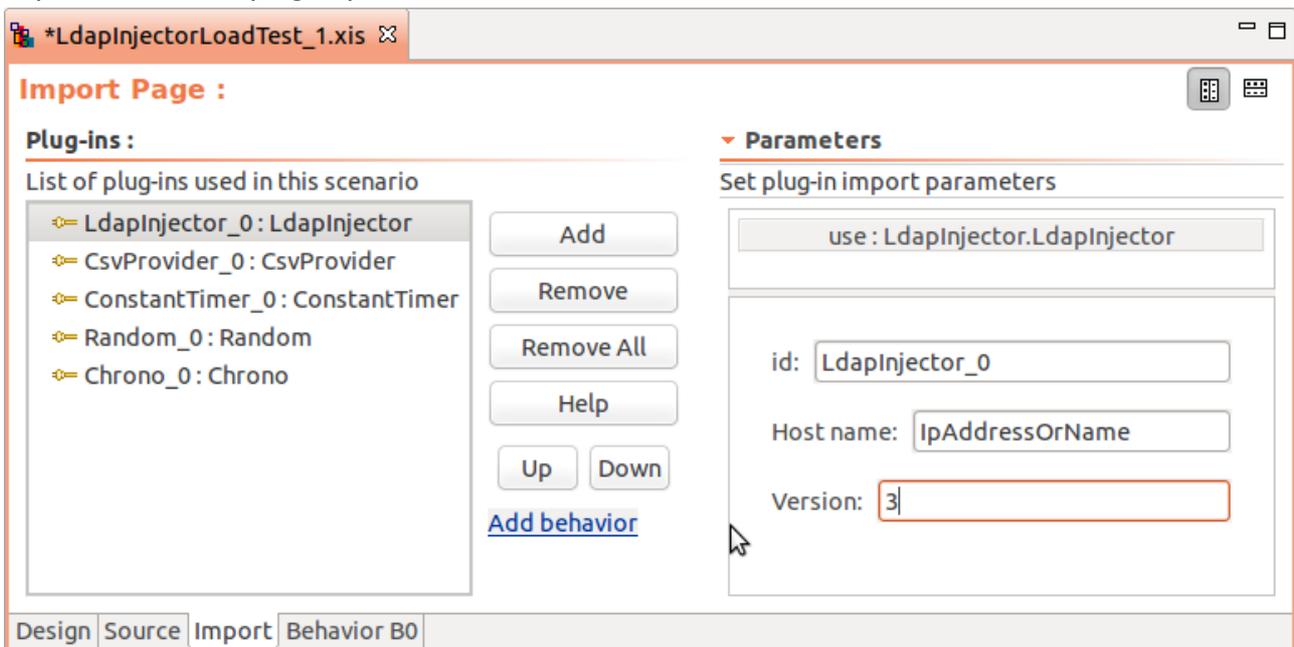


## How To Use CLIF v2 and ISAC plug-ins

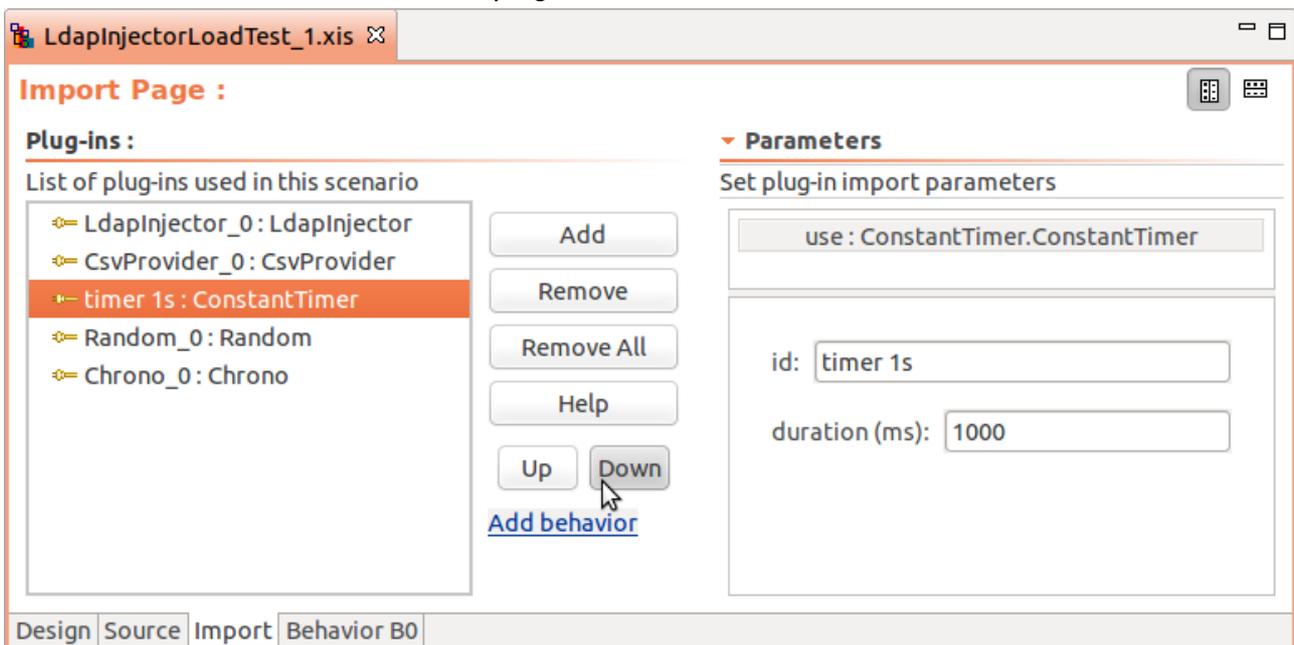
Now all the necessary plug-ins are listed in Import tab.



Then, we must set the plug-ins' initialization parameters. Select the LdapInjector\_0:MyLdapInjector import and set the plug-in parameters:



Do the same with the ConstantTimer plug-in:



Note: each import has a unique identifier (see the "id" field). You may keep the default generated id (*PlugInName\_0*, *PlugInName\_1*, etc.), but most of the time, giving a shorter and more explicit id is convenient. For example, here we can give the id "timer 1s" for the ConstantTimer import initialized with a 1000ms duration.

The Random and Chrono plug-ins have no parameter.

## How To Use CLIF v2 and ISAC plug-ins

Select the CSVProvider plug-in and set its parameters:

use : CsvProvider.CsvProvider	
id:	csv
File name:	LdapLoadTest_1.csv
Field separator:	#
fields names (separated by the given separator):	login#password#searchBase#searchFilter#searchScope
MacOS9 line separator	<input type="checkbox"/> enable
shared	<input type="checkbox"/> enable
loop	<input checked="" type="checkbox"/> enable

The file name is the name of the CSV file that will contain the external data set for the LDAP load injection. A line in the CSV File will have this format:

```
login#password#searchBase#searchFilter#searchScope
uid=0,ou=appli,dc=annuaire,dc=com#secret_0#ou=appli,dc=annuaire,dc=com#(&(uid=0)(autorisation=all))#2
uid=1,ou=appli,dc=annuaire,dc=com#secret_1#ou=appli,dc=annuaire,dc=com#(&(uid=1)(autorisation=all))#2
uid=2,ou=appli,dc=annuaire,dc=com#secret_2#ou=appli,dc=annuaire,dc=com#(&(uid=2)(autorisation=all))#2
```

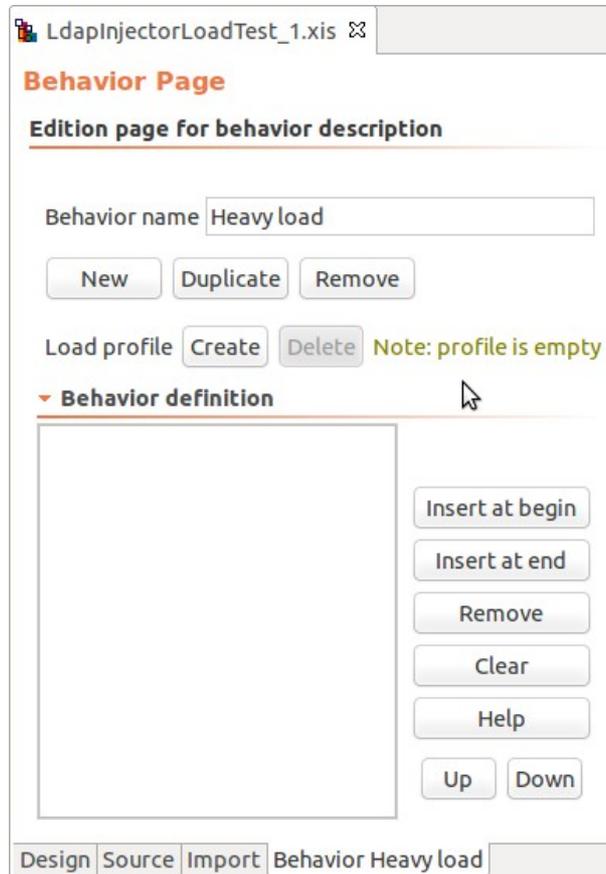
In the "Field separator" field, you have to set the separator character used in the CSV file. (# in our case). In the "Fields names" field, enter the list of the parameters names (columns) in the CSV file separated with the given separator (# here).

Description of the check-boxes:

- MacOS9 line separator: use CR instead of LF as line separator.
- shared: when set, progression in the lines is shared by all session objects. In other words, each session object will get a different line instead of all getting the same sequence of lines.
- loop: when set, the line sequence wraps up to the first line when the end of file has been reached. Otherwise, an alarm is thrown when trying to get a field value while the end of file has been reached, and the empty string is used as value.

### 2.2.3. Define your behavior

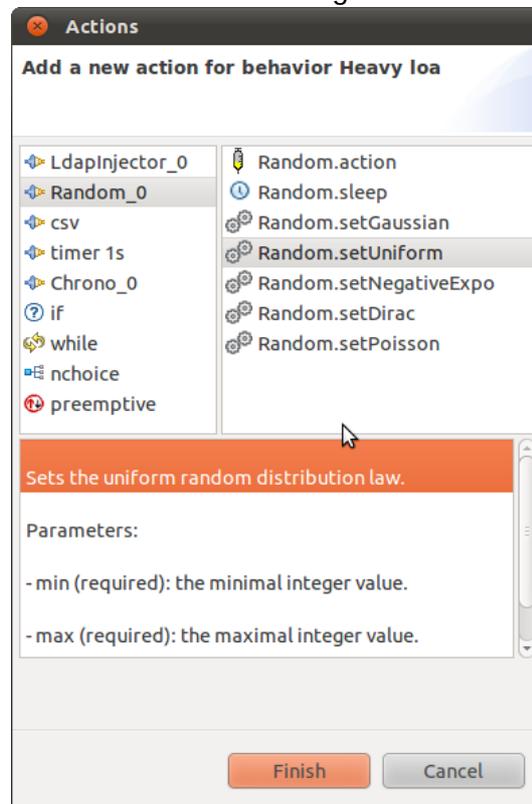
To define a virtual user behavior, you have to select the Behavior tab of your ISAC scenario file. You get the behavior editor. You can now change the default behavior name ("B0") if you wish. Here, we change it for "Heavy load").



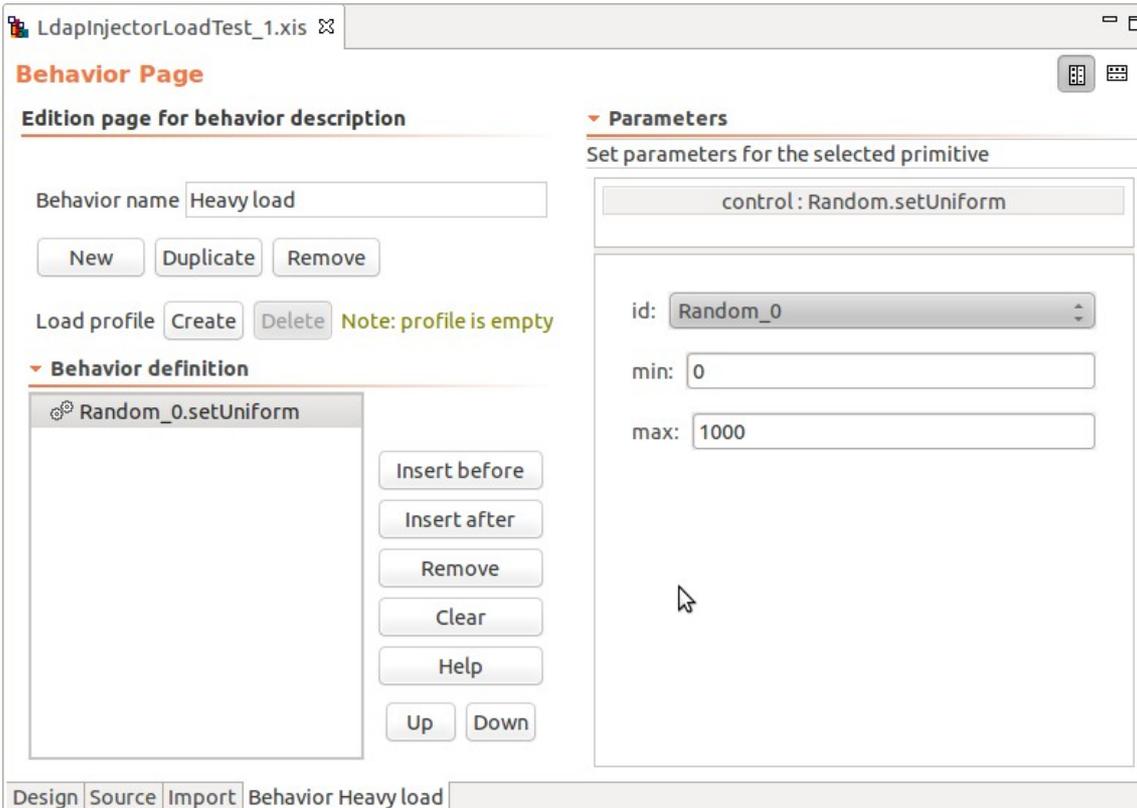
## How To Use CLIF v2 and ISAC plug-ins

At the beginning of the behavior, we add a sleep action. The sleep time will be variable and its variation will depend on a uniform random distribution.

Click on the "Insert at begin" button, choose the Random\_0 import on the left-hand side of the pop-up window, and select its setUniform control in the right hand-side. Click on Finish.

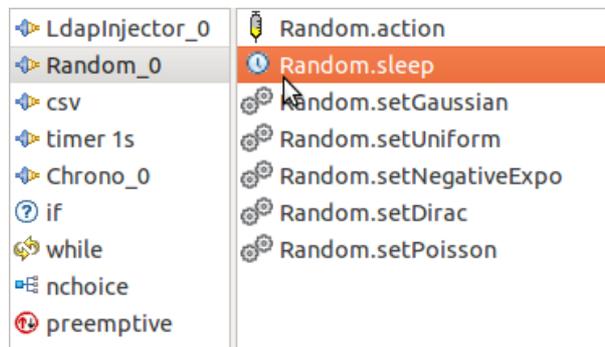


We set the required parameters' values:



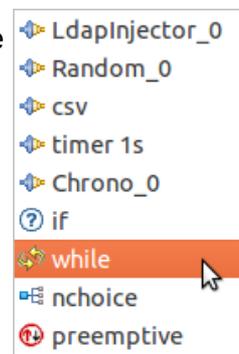
And now, we add a timer that will make the behavior have a sleep/think time for a random duration uniformly distributed between 0 and 1000ms.

Click on the "Insert after" button, choose the Random\_0 import and select the sleep timer primitive.



This timer primitive takes no parameter.

Then, we add a while loop. Click on the "Insert after" button and choose the while statement in the pop-up window.



## How To Use CLIF v2 and ISAC plug-ins

The loop condition is: while we are not at the end of the CSV file, we iterate, so that each virtual user uses all the lines from the CSV file, and then terminates its execution.

Select the `while` statement in the behavior editor and choose the `CsvProvider.notEndOfFile` condition among the conditions provided by the different imported plug-ins.

The screenshot shows the 'Behavior Page' for editing a behavior named 'Heavy load'. The page is divided into several sections:

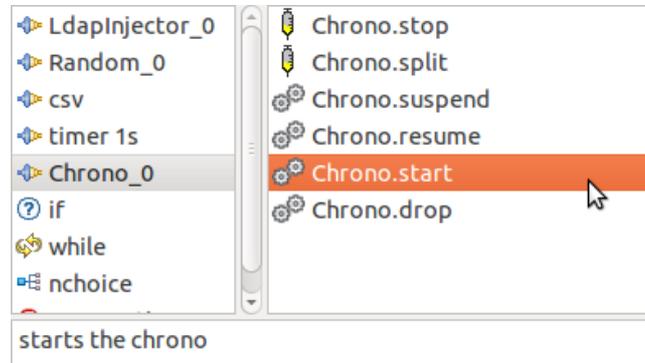
- Behavior name:** 'Heavy load' (input field)
- Buttons:** 'New', 'Duplicate', 'Remove' (for behavior name); 'Create', 'Delete' (for load profile); 'Add child', 'Insert before', 'Insert after', 'Remove', 'Clear', 'Help', 'Up', 'Down' (for behavior definition)
- Load profile:** 'Create', 'Delete' (Note: profile is empty)
- Behavior definition:** A list of actions: 'Random\_0.setUniform', 'Random\_0.sleep', and 'while' (selected).
- Parameters:** 'Set parameters for the selected primitive'. A dropdown menu shows 'Select a test condition of the while controller node:' with 'CsvProvider.notEndOfFile' selected. Below it, an 'id:' dropdown shows 'csv' selected.

At the bottom, there are tabs for 'Design', 'Source', 'Import', and 'Behavior Heavy load'.

Note: the "id" parameter must be checked carefully when the same plug-in has been imported more than once, to make the difference between the imports. Here, there is only one possible choice ("csv" plug-in id).

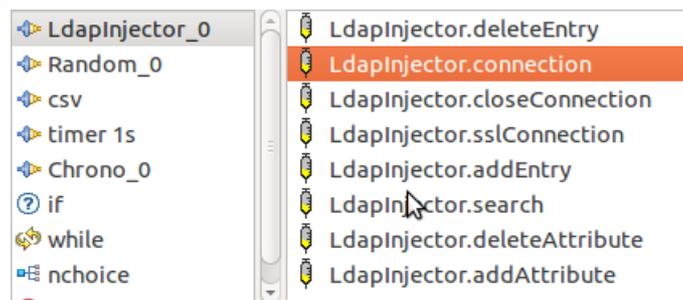
To include actions in the loop, select the while statement in the Behavior definition and click on the "Add child" button.

The first action that we will add is starting chronometer Chrono\_0 with its start control. This control takes no argument.



Then, we perform a connection to the LDAP directory. Select the Chrono\_0.start instruction, and click on the "Insert after" button. Alternatively, you may select the while statement and click on the "Add child" button.

In the action selector, get the connection action from the LdapInjector\_0 plug-in.



## How To Use CLIF v2 and ISAC plug-ins

Now, we have to set the LDAP connection parameters' values. These values are provided by the CSV file, through variable statements `${pluginId:key}`. These statements will be replaced at runtime by the value associated to `key` (here, `login` or `password`) by the plug-in referenced by `pluginId` (the plug-in import id, namely "csv" in this case).

The screenshot displays the 'Behavior Page' for editing a behavior named 'Heavy load'. The interface is split into two main areas:

- Edition page for behavior description:** This section contains a 'Behavior name' field with the value 'Heavy load'. Below it are buttons for 'New', 'Duplicate', and 'Remove'. The 'Load profile' section has 'Create' and 'Delete' buttons, with a note that says 'Note: profile is empty'. The 'Behavior definition' section shows a tree view with 'LdapInjector\_0.connection' selected. To the right of this tree are buttons for 'Insert before', 'Insert after', 'Remove', 'Clear', 'Help', 'Up', and 'Down'.
- Parameters:** This section is titled 'Set parameters for the selected primitive'. It shows a dropdown menu for 'sample : LdapInjector.connection'. Below this, there are fields for 'id' (set to 'LdapInjector\_0'), 'Ldap connection port' (set to '389'), 'Login DN' (set to `${csv:login}`), and 'Password' (set to `${csv:password}`).

At the bottom of the window, there is a navigation bar with tabs for 'Design', 'Source', 'Import', and 'Behavior Heavy load'.

Note: these variable statements are not specific to the `CsvProvider` plug-in. Many other plug-ins provide helpful variables. Refer to the plug-ins Help from the Import tab, or see the ISAC plug-ins reference manual.

Then, we add an action to perform a search on the LDAP directory. Use the same method as for the LDAP connection action to add the `LdapInjector_0.search` request, and select it in the behavior editor to set its parameters:

The screenshot shows the 'Behavior Page' for a test named 'LdapInjectorLoadTest\_1.xis'. The page is divided into two main sections: 'Edition page for behavior description' and 'Parameters'.

**Edition page for behavior description:**

- Behavior name:
- Buttons: New, Duplicate, Remove
- Load profile:   Note: profile is empty
- Behavior definition:**
  - Random\_0.setUniform
  - Random\_0.sleep
  - while
    - Chrono\_0.start
    - LdapInjector\_0.connection
    - LdapInjector\_0.search** (selected)
- Buttons: Insert before, Insert after, Remove, Clear, Help, Up, Down

**Parameters:**

Set parameters for the selected primitive

sample : LdapInjector.search

id:

Search base:

Search Filter:

Search scope:

Design Source Import Behavior Heavy load

## How To Use CLIF v2 and ISAC plug-ins

Then, we close the connection to the LDAP directory. Add action `LdapInjector_0.closeConnection` to the while loop. It takes no parameter.

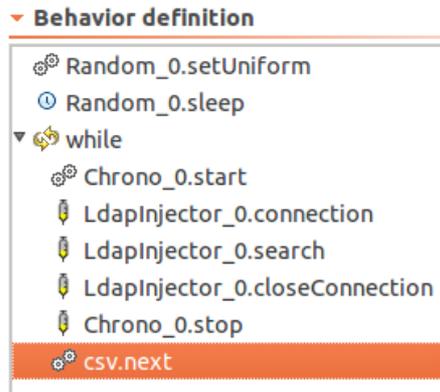
Next, add the `Chrono_0.stop` instruction to the while loop, to get the total duration of each connection-search-disconnection transaction. Select this instruction in the behavior editor and set its parameters. This instruction creates a pseudo request report, with the full execution time of the transaction. You have to state if the transaction must be considered as successful or not (see the status check-box). Optionally, you may add a specific comment and a result string in the corresponding fields.

The screenshot shows the 'Behavior Page' for 'LdapInjectorLoadTest\_1.xis'. The page is titled 'Edition page for behavior description' and contains the following elements:

- Behavior name:** Heavy load
- Buttons:** New, Duplicate, Remove
- Load profile:** Create, Delete (Note: profile is empty)
- Behavior definition:** A tree view showing the following actions:
  - Random\_0.setUniform
  - Random\_0.sleep
  - while
    - Chrono\_0.start
    - LdapInjector\_0.connection
    - LdapInjector\_0.search
    - LdapInjector\_0.closeConnection
    - Chrono\_0.stop** (selected)
- Parameters:** Set parameters for the selected primitive (sample : Chrono.stop)
  - id: Chrono\_0
  - status:  successful
  - comment (optional): search transaction
  - result (optional):
- Action buttons:** Insert before, Insert after, Remove, Clear, Help, Up, Down

The bottom of the window shows a tabbed interface with 'Design', 'Source', 'Import', and 'Behavior Heavy load' tabs.

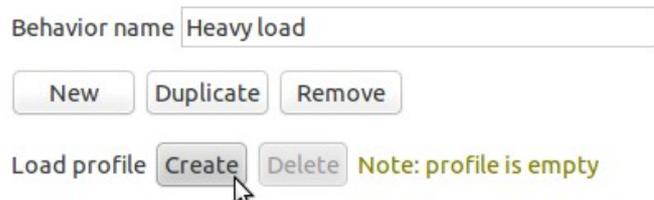
Finally, end the while loop block with the `csv.next` control instruction to get the next line from the CSV file. The behavior is now complete, and looks like this:



### 2.2.4. Load Profiles

Load profiles enable predefining how the population of each behavior will evolve, by setting the number of active instances according to time.

#### Edition page for behavior description

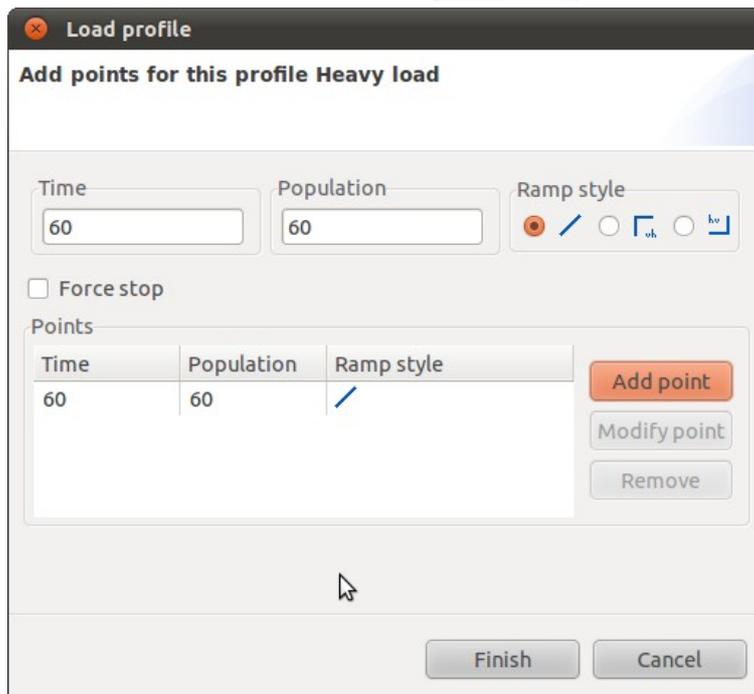


To create a load profile, select the corresponding Behavior tab and click on the "Create" button.

First, we want a ramp that increases the number of simultaneously active behavior instances (virtual users) from 0 to 60 during the first minute (60 seconds):

- enter 60 in the Time field
- enter 60 in the Population field
- choose the linear Ramp style

To save these settings click on the Add point button: 



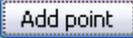
## How To Use CLIF v2 and ISAC plug-ins

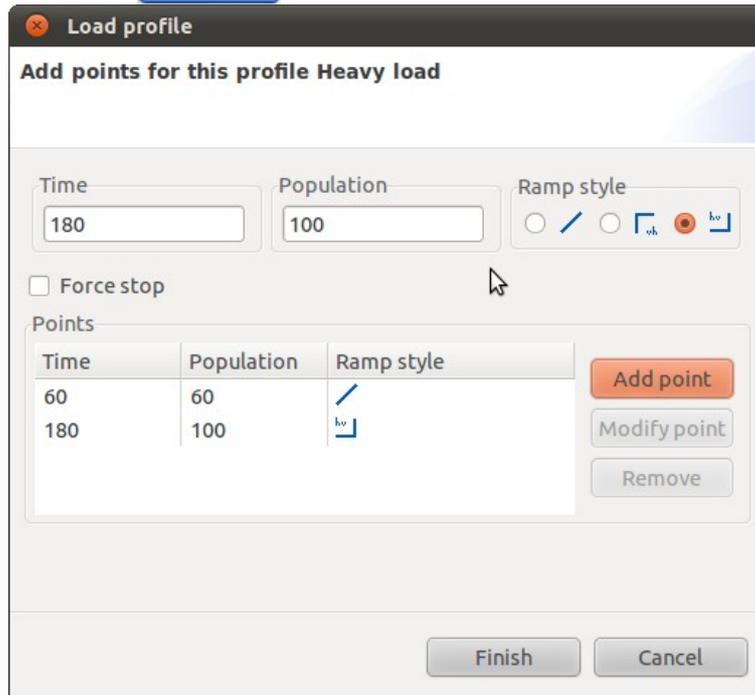
Note: point (0,0) is implicit and will be automatically inserted whenever no point is specified for time 0.

Then, we want a constant number of simultaneously active behavior instances during 2 minutes (120 seconds). Since the "Time" box means the time in seconds since the start of the test, the new point to add will be at 120 (2 minutes) + 60 (initial ramp-up) = 180 seconds. Next, we want the number of virtual users to rise to 100 as a burst.

Enter 180 in the Time field and 100 in the Population field. Choose the "horizontal-vertical crenel" (hv) ramp style:



Click on the Add point button: 



**Load profile**  
Add points for this profile Heavy load

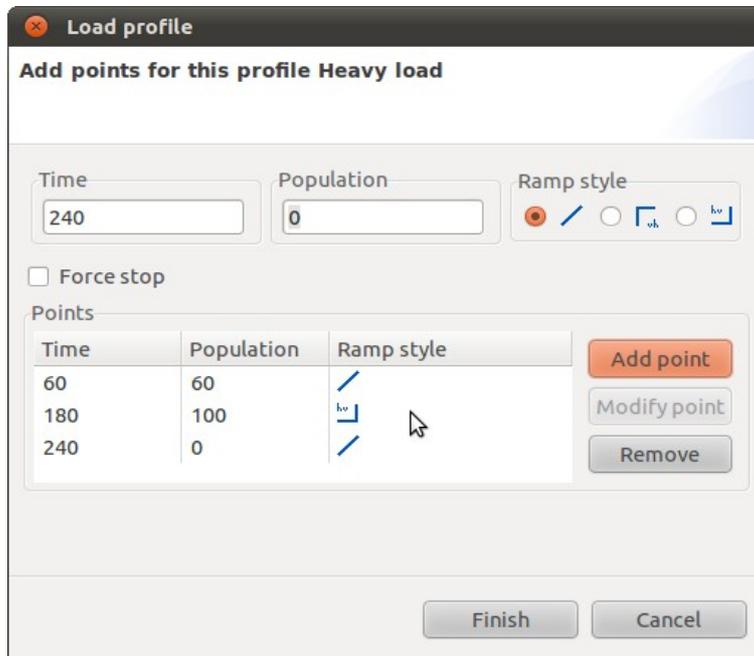
Time:  Population:  Ramp style:

Force stop

Time	Population	Ramp style
60	60	
180	100	

Finally, we want to decrease the number of virtual users from 100 to 0 during 60 seconds.

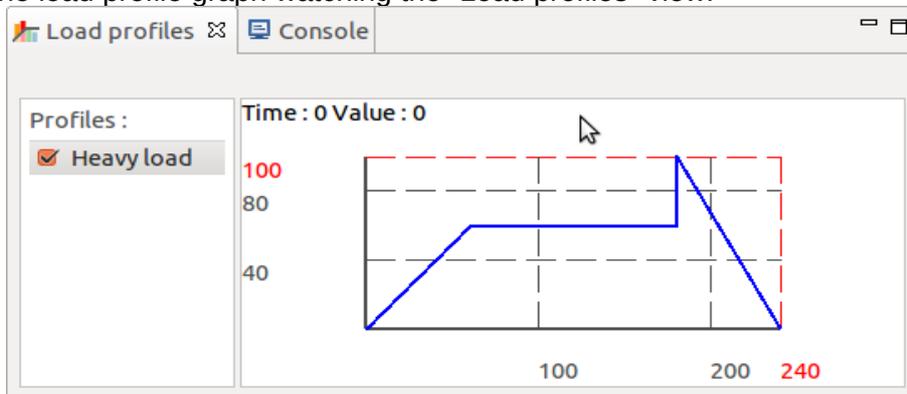
Enter 240 in the Time field and 0 in the Population field, and select the linear Ramp style. click on the Add point button:



Note: the "Force stop" flag states if some active instances shall be stopped to enforce a decrease of the population according to the load profile, or if the extra behaviors shall complete.

Click on the Finish button:

You can see the load profile graph watching the "Load profiles" view:



Note: the end of load profile implicitly means an instant drop to 0 virtual users (this is explicit here since we defined a ramp down to 0 at time 240s).

## 2.3. Define the test plan

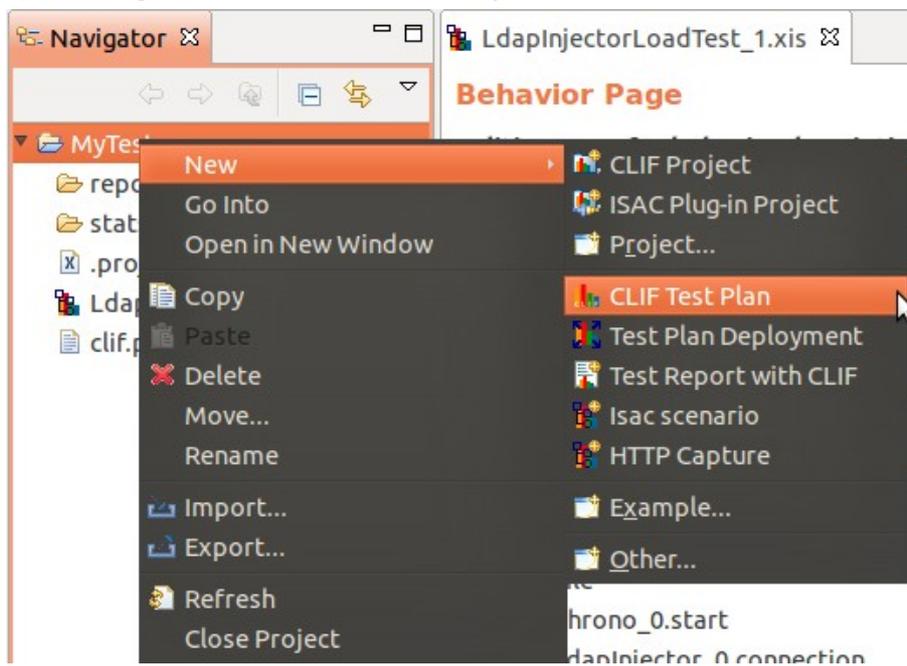
### 2.3.1. Rationale

We want to test an LDAP directory. To test it and to get performance measures we will deploy injectors and probes. To have a powerful load injection, we want to send requests from two different servers. To be able to check the injection servers' CPU usage, we will deploy on each of them a CPU probe. To be able to monitor the CPU usage on the LDAP server, we will deploy a CPU probe on it.

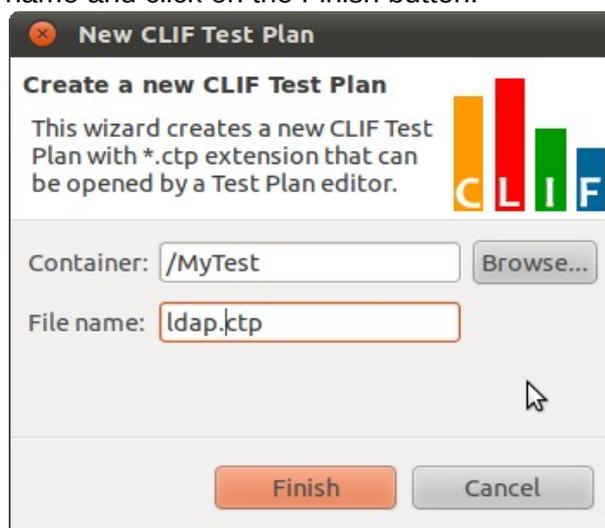
### 2.3.2. Create a test plan

This test plan will be created at the same location of the ISAC scenario file.

Select project MyTest, right-click on it and choose option "New>CLIF Test Plan"



Then enter your test plan name and click on the Finish button:

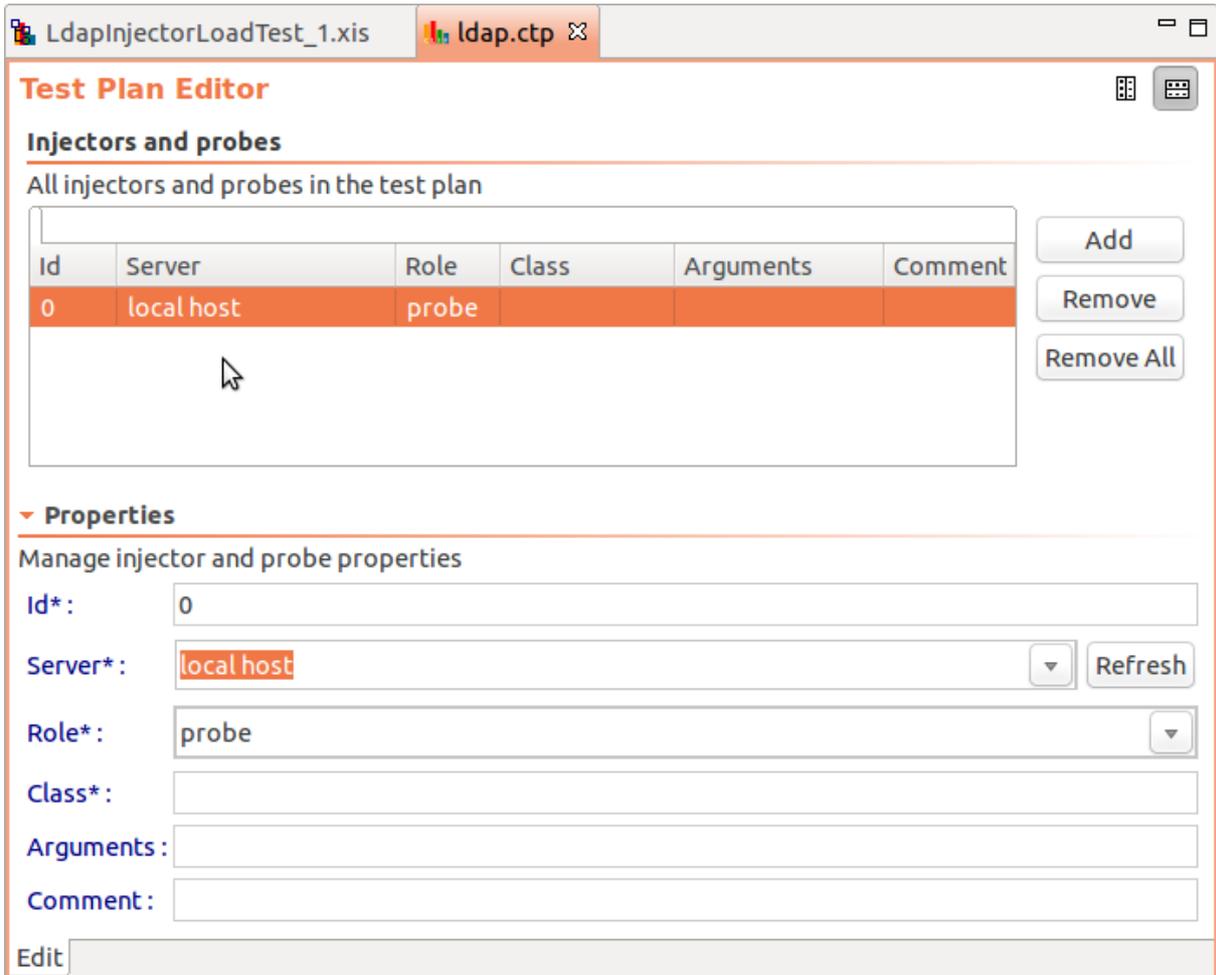


### 2.3.3. Add Probes and Injectors to the test plan

In our case we will add:

- Two injectors, one on each server used to inject requests on the LDAP directory.
- Three CPU probes, one on each load injection server and one on the server where the LDAP directory is installed.

Click on the add button  and set the properties of the injector or probe:



**Test Plan Editor**

**Injectors and probes**

All injectors and probes in the test plan

Id	Server	Role	Class	Arguments	Comment
0	local host	probe			

**Properties**

Manage injector and probe properties

Id\*: 0

Server\*: local host

Role\*: probe

Class\*:

Arguments:

Comment:

Edit

As we want to deploy load injectors and probes on remote machines, we must be able to define CLIF servers other than the default "local host" (which is embedded in the console). To be able to change the Server name, you have to run the target CLIF servers once the registry is started in the console (see section 4), and click on the "Refresh" button.

```
Id: 0
Server: inj1
Role: probe
Class: cpu
Arguments: 1000 600 (a measure will be taken each 1000 ms and during 600 s)
```

Click on the Add button.

Note: the injector or probe Id field has a default integer value (0, 1, 2, etc.). However, you might change it and use an arbitrary, and possibly more explicit, string. Should you change the Id, be very careful about using unique Ids. Using the same Id for different injectors or probes in the same test plan will result in weird behaviors.

## How To Use CLIF v2 and ISAC plug-ins

```
Id: 1
Server: inj2
Role: probe
Class: cpu
Arguments: 1000 600
```

Click on the Add button.

```
Id: 2
Server: sut
Role: probe
Class: cpu
Arguments: 1000 600
```

Don't forget to save your test plan clicking on File -> Save (or Ctrl+S)

Now we have this view:

**Injectors and probes**

All injectors and probes in the test plan

cpu					
Id	Server	Role	Class	Arguments	Comment
2	sut	probe	cpu	1000 600	System Under Test CPU load
1	inj2	probe	cpu	1000 600	CPU load of injector 2
0	inj1	probe	cpu	1000 600	CPU load of injector 1

And finally, add injectors on both CLIF servers inj1 and inj2. Since we are using an ISAC scenario, IsacRunner must be used as injector class, and the scenario file name must be given as argument. Don't forget to select the "injector" as Role.

Click on the Add button.

```
Id: 3
Server: inj1
Role: injector
Class: IsacRunner
Arguments: LdapInjectorLoadTest_1.xis
```

Click on the Add button.

```
Id: 4
Server: inj2
Role: injector
Class: IsacRunner
Arguments: LdapInjectorLoadTest_1.xis
```

Note: by deploying twice the same ISAC scenario, the overall generated traffic will be double. In other words, the global load profile will be twice as big as the

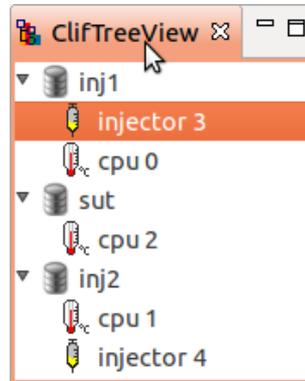
Now we have this view:

**Injectors and probes**

All injectors and probes in the test plan

cpu injector					
Id	Server	Role	Class	Arguments	Comment
3	inj1	injector	IsacRunner	LdapInjectorLoadTest_1.xis	LDAP load injector
4	inj2	injector	IsacRunner	LdapInjectorLoadTest_1.xis	LDAP load injector

The ClifTree View gives a complementary view on this test plan, organized by CLIF server. You may get this view from menu "Window>Views", or by calling the CLIF perspective from menu Window>Perspectives. Copy-paste operations are possible in this view.

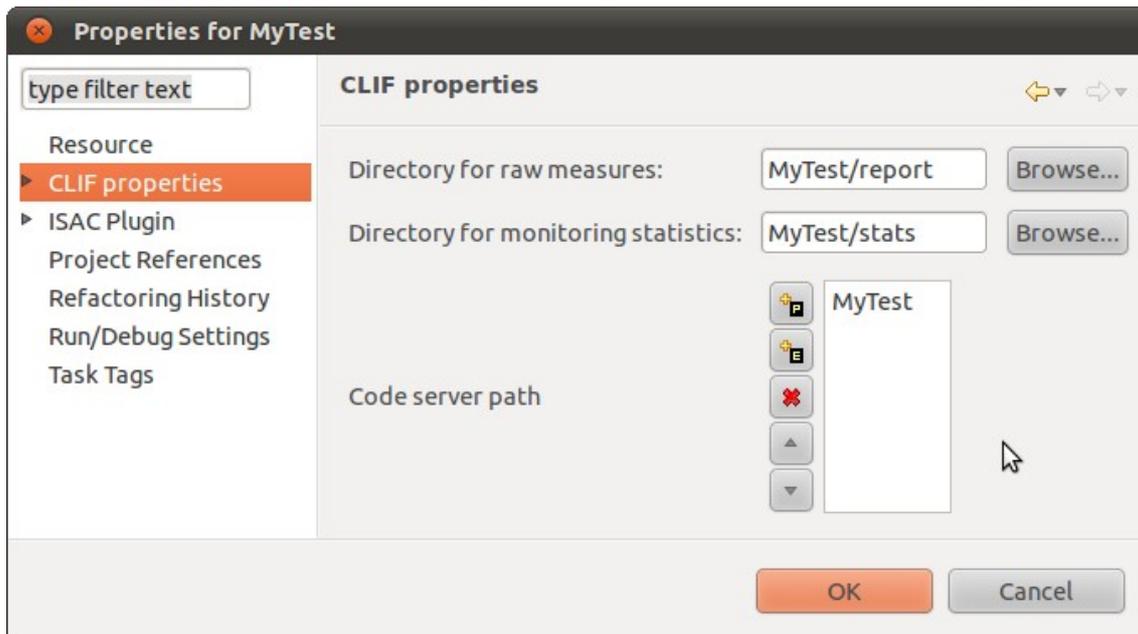


## 2.4. Deploying and executing the test plan

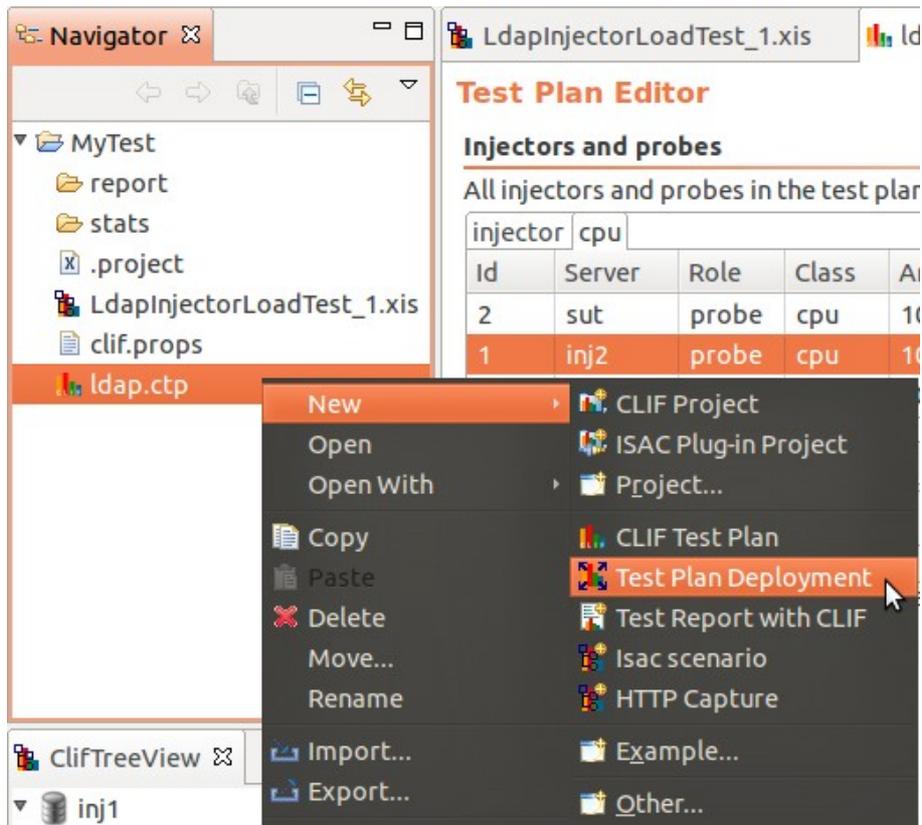
To ease the deployment of distributed test plans, CLIF embeds a "code server" feature which transparently ships resource files used by your test plans and scenarios: the ISAC scenarios files, as well as possible files used by the scenario files, such as the CSV file we use here to set LDAP connection and search parameters.

In order to find these files, the code server must be given a resolution path. With the Eclipse-based console, this path is set by default to the current CLIF project path. You may check and modify this path from the "CLIF properties" tab available among the project properties.

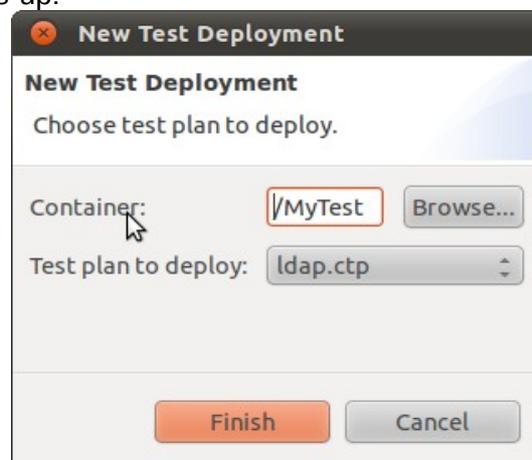
Right-click on the project and choose "Properties" option. Then, select the CLIF properties line.



Now, right-click on the MyTest project in the Navigator view (alternatively, use the File menu), and run option New>Test Plan Deployment.

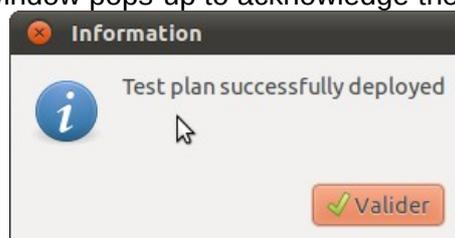


The deployment wizard pops-up:



Check that the "Container" field's value is /MyTest. Otherwise, click on the "Browse..." button and select the MyTest project.

Click on Finish. A information window pops-up to acknowledge the deployment success.



## How To Use CLIF v2 and ISAC plug-ins

Now, the ldap.ctp editor displays a new "Test" tab (the tab switch is at the bottom part: Edit, Test). It contains the same view on the test plan as the Edit tab, but:

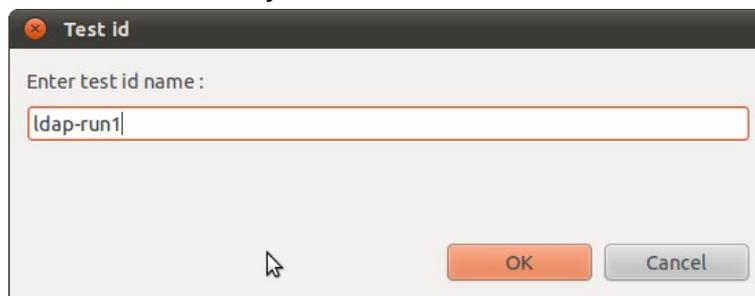
- it is not editable,
- buttons have appeared: 6 at the bottom, and 2 on the right-hand side
- a global status is displayed on the right-hand side ("deployed")

The screenshot shows the 'Test Commands' window in the ldap.ctp editor. The window title is 'LdapInjectorLoadTest\_1.xis' and 'ldap.ctp'. The main section is titled 'Injectors and probes' and contains the text 'All injectors and probes in the test plan'. Below this is a table with columns: injector, cpu, Id, Server, Role, Class, Arguments, Comment, and State. The table contains three rows of data. To the right of the table are buttons for 'Select All' and 'Deselect All', and a 'Global state: deployed' indicator. Below the table are buttons for 'Initialize', 'Start', 'Suspend', 'Stop', 'Collect', and 'Parameters'. At the bottom, there are tabs for 'Edit' and 'Test'.

injector	cpu	Id	Server	Role	Class	Arguments	Comment	State
2	sut	2	sut	probe	cpu	1000 600	System Under Test CPU load	deployed
1	inj2	1	inj2	probe	cpu	1000 600	CPU load of injector 2	deployed
0	inj1	0	inj1	probe	cpu	1000 600	CPU load of injector 1	deployed

Before starting the test execution, the deployed test plan must be initialized.

Click on the Initialize button. A dialog window pops-up to ask for a name for this test run. The default name is the test plan file name (without the .ctp extension). You may modify this name as you wish. In any case, the initialization date will be appended to this test name, and the result will be used to create the measures directory name.



As soon as the test is initialized, the Monitor view appears. This view contains a set of tabbed panels:

- one for all injectors
- one for each probe class (here, just cpu)

The screenshot displays the IsacRunner interface. The top window, titled 'LdapinjectorLoadTest\_1.xis', shows the 'Test Commands' panel. Under 'Injectors and probes', a table lists three items:

injector	cpu						
Id	Server	Role	Class	Arguments	Comment	State	
<input checked="" type="checkbox"/>	2	sut	probe	cpu	1000 600	System Under Test CPU load	initialized
<input checked="" type="checkbox"/>	1	inj2	probe	cpu	1000 600	CPU load of injector 2	initialized
<input checked="" type="checkbox"/>	0	inj1	probe	cpu	1000 600	CPU load of injector 1	initialized

Buttons for 'Initialize', 'Start', 'Suspend', 'Stop', 'Collect', and 'Parameters' are visible. The 'Global state' is 'initialized'. Below this is the 'Monitor' panel, showing a graph of CPU usage over time. The graph has a y-axis labeled '%CPU' ranging from 0 to 14 and an x-axis showing time from 01:21:40 to 04:10. A red dashed line is at 14, and a blue line shows the CPU usage fluctuating between approximately 2 and 12. A tooltip shows 'Time: 60 Value: 9'. A table on the left of the graph lists injectors with checkboxes for 'Show' and 'Listen':

Show	Listen	Id	Time
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	2	00"
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	00"
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	00"

At the bottom of the monitor panel, there are input fields for 'Drawing time frame (s)' (300) and 'Polling period (s)' (1), along with 'Refresh' and 'Reset' buttons.

The user may set the monitoring time-frame (, the polling period (time interval between two consecutive monitoring points), and start or stop the monitoring process. Moreover, a check-box table at the left side of each panel makes it possible to selectively disable or enable the collect and display of monitoring data, for each blade.

Now you can start the injection by clicking on the Start button. Then you can suspend/resume, or stop the test execution by clicking on the corresponding buttons. The Parameters button allows for changing some probe/injector-specific parameters. With IsacRunner injectors, several parameters may be modified at runtime:

- about the scenario execution engine itself (size of the thread pool, tolerance on deadlines, storage options...);
- number of virtual users (aka population) of each defined behavior.

### 3. Browsing test results

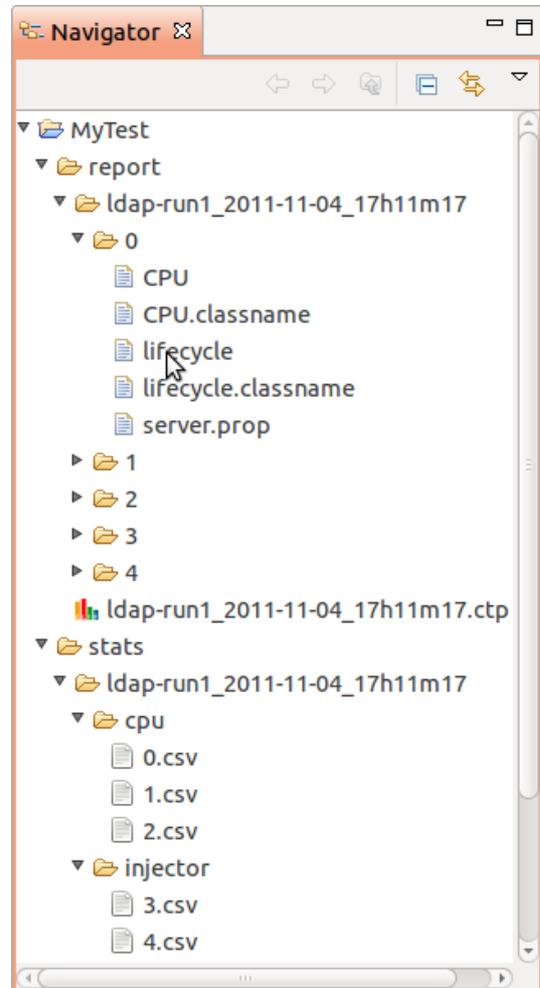
#### 3.1. Getting raw measures

Once the test is terminated (naturally completed or manually stopped), you shall collect raw measures produced by injectors and probes by clicking on the Collect button, unless you are not interested in these measures. Collecting measures will locally create CSV formatted files containing all the measures, in a tree-organized breakdown of directories and files.

The root directory is determined by a CLIF property. The default is a directory named "report" in the corresponding CLIF project. This directory contains one sub-directory per initialized test, whose name is the concatenation of the chosen test id and the initialization date. Each sub-directory has a companion .ctp file, which is a copy of the deployed test plan.

For each test run, each injector and probe in the test plan has its own sub-directory, whose name is the injector/probe Id. It contains a number of text files:

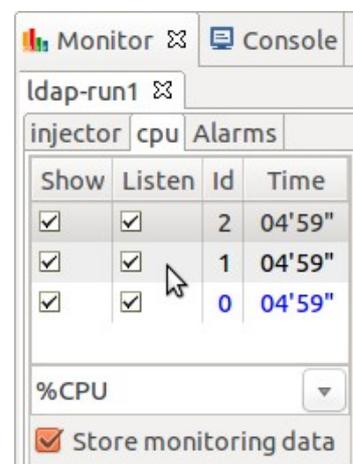
- lifecycle lists the dates of the execution status changes;
- server.prop lists all the Java system properties of the Java Virtual Machine running this injector or probe;
- alarm contains the alarm events thrown by this injector or probe, if any;
- for injectors only, file action contains the request execution reports (duration, success...);
- for probes only, a specific file, whose name and content depends on the probe class, contains all the performed measures.



#### 3.2. Getting monitoring data

Besides the raw measures, monitoring data can optionally be stored also, instead of just being displayed and discarded. Monitoring data provide moving statistics that are very convenient to get a quick analysis of the test results. To enable this feature, mark the "Store monitoring data" check-box for each class of probe/injector you are interested (here, injector and cpu for example).

Then, monitoring data is written to a set of tree-organized breakdown of directories and CSV formatted files. The root directory of this tree is determined by a CLIF property. The default is a directory named "stats" in the corresponding CLIF project. This directory contains one sub-directory per initialized test, whose name is the same as the corresponding raw measure sub-directory. It contains one sub-directory for injectors and one sub-directory per probe class. Monitoring data for each injector and probe is stored in the relevant directory, in a file whose name is *Id*.csv.



## 4. CLIF servers and registry

### 4.1. Installation

Refer to the Installation Guide.

### 4.2. Rationale

CLIF servers are necessary to deploy any test plan, since they host load injectors and probes. CLIF servers are designated by a name, which is registered in a Registry. In order to run, CLIF servers must be able to find this Registry, which implies that:

- the Registry must be running before a CLIF server can be launched;
- parameters must be given to tell the CLIF servers where to find the Registry and register themselves.

### 4.3. Running a registry

The CLIF registry is launched when necessary by the Java Swing GUI or the eclipse-based GUI. If you want to launch it from command-line, see the User Manual.

Here, we will be using the Eclipse-based console GUI. Opening a CLIF test plan (.ctp file) will automatically run the Registry (unless a registry is already running). To check that the Registry is running, you may have a look at the "Console View" and look for a message of this kind:

```
Creating a CLIF Registry... Fractal registry is ready.
registry@localhost:1234
```

### 4.4. Configuring a CLIF server

You may configure CLIF either by editing file `clif.props` in the `etc/` subdirectory, or by using command `"ant config"` (refer to the section dedicated to the command line interface in the User Manual). This configuration operation must be done everywhere you want to run a CLIF server. You may also make this configuration step only once, and copy the resulting file `etc/clif.props` wherever needed.

### 4.5. Running a CLIF server

Once CLIF is configured, and before running CLIF servers, ensure the CLIF registry is running. Trying to run a CLIF server, either when the Registry is not running, or with an wrong configuration, will result in a CLIF server failure, with this "Connection refused" message:

```
[java] Exception in thread "main" org.objectweb.fractal.rmi.RemoteException: error
during marshalling/unmarshalling by stub:
org.objectweb.jonathan.apis.presentation.MarshalException: exception preparing
marshaller: java.net.ConnectException: Connection refused
[java]         at
org.objectweb.fractal.rmi.stub.Stub.handleException(Stub.java:292)
[java]         at
org.objectweb.fractal.rmi.registry.NamingService_Stub.list(Unknown Source)
[java]         at org.ow2.clif.deploy.ClifRegistry.<init>(ClifRegistry.java:95)
[java]         at
org.ow2.clif.server.lib.ClifServerImpl.main(ClifServerImpl.java:139)
```

In our case we will deploy three CLIF servers, with names `inj1`, `inj2` and `sut`, by running these commands:

## How To Use CLIF v2 and ISAC plug-ins

- `ant -Dserver.name=inj1 server`
- `ant -Dserver.name=inj2 server`
- `ant -Dserver.name=sut server`

Of course, these CLIF servers should be created on different computers, but it is possible to create as many CLIF servers as you like on the same computer. In any case, CLIF server names must be unique for a given Registry.