March, 19th 2013

CLIFv2 Quick Start manual

http://clif.ow2.org/

Table of contents

March, 19th 2013

1 Introduction

In this Quick Start, you will learn how to create your first CLIF project.

1.1 Requirements

We will use the Eclipse-based CLIF console in this Quick Start. Refer to the Installation Manual.

1.2 Summary

We will make three different tests. In the first test, you will learn how to create a CLIF project, a Test Plan, add probes to the Test Plan and deploy it.

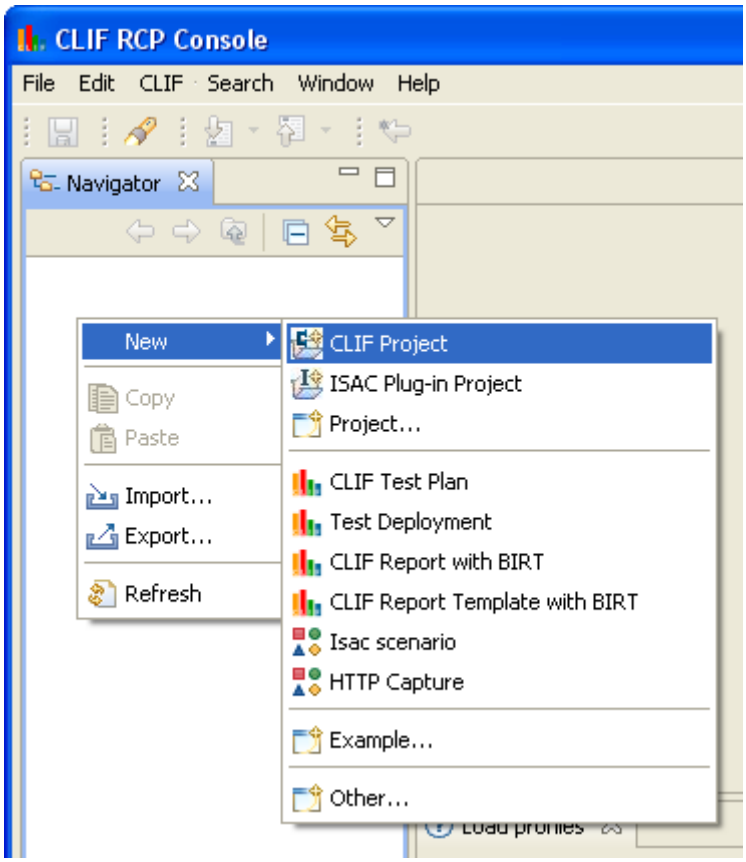
In the second test, we will create an Isac Scenario using several plug-ins to test an HTTP server.

In the third test, we will use the HTTP Capture wizard to create a scenario and replay it.

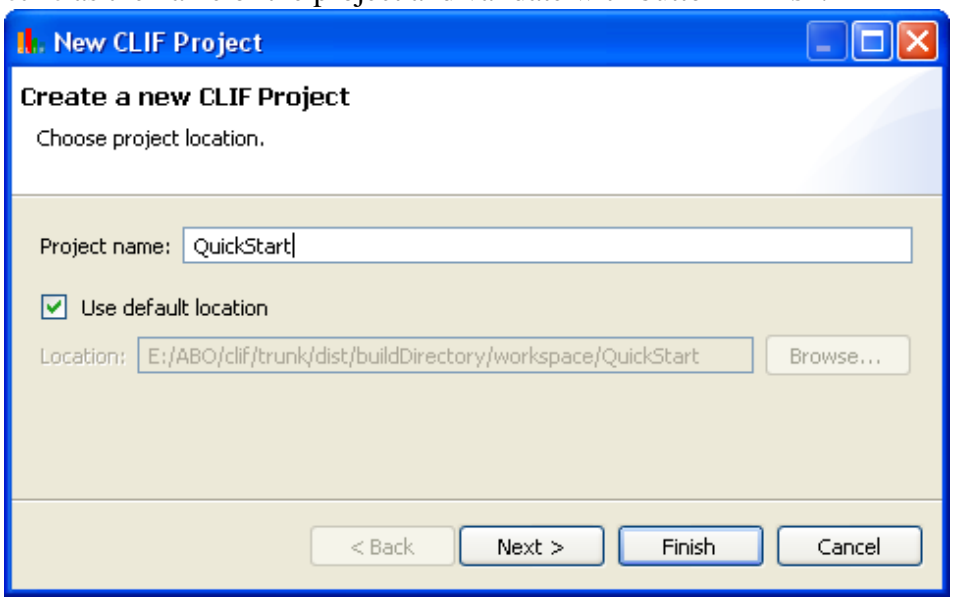
2 Basic test

2.1 Project creation

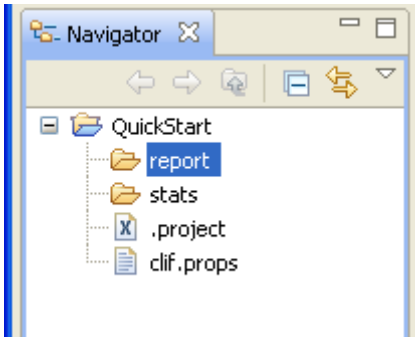
Create a new CLIF project with wizard "New → CLIF Project"



Enter `QuickStart` as the name of the project and validate with button `Finish`.



You will have a CLIF skeleton project created with base files.

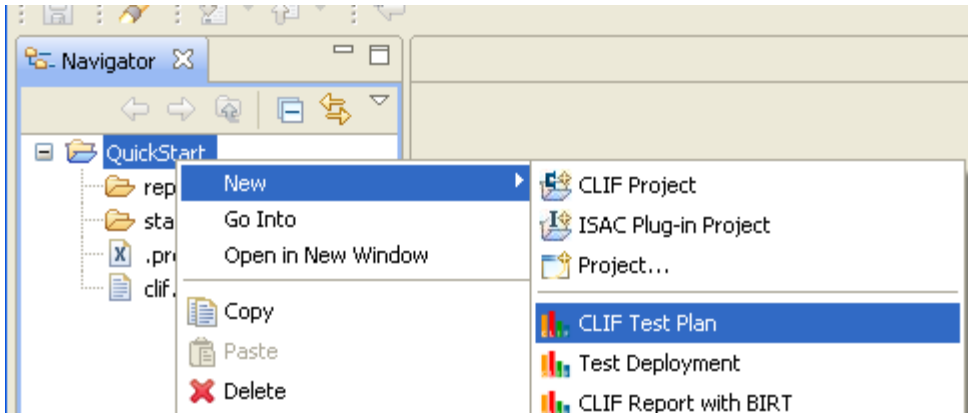


File `clif.props` contains specific configuration properties for the project. See the User Manual for more details about CLIF's properties. The `report` directory will contain test results.

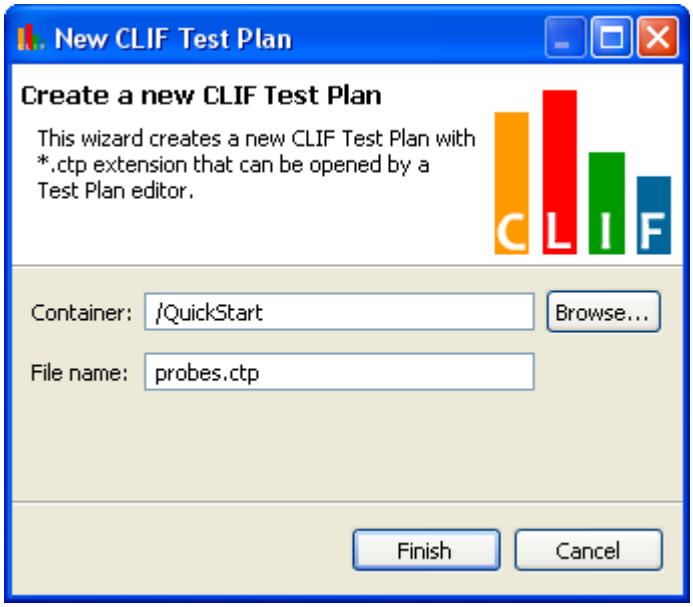
This project will act as your *container* for your test plan and scenario files. A test plan is a list of probes and injectors deployment definitions. As a first touch with CLIF, we create a basic test plan with a single probe.

2.2 Test Plan Creation

Create a Test Plan with wizard "New → CLIF Test Plan"



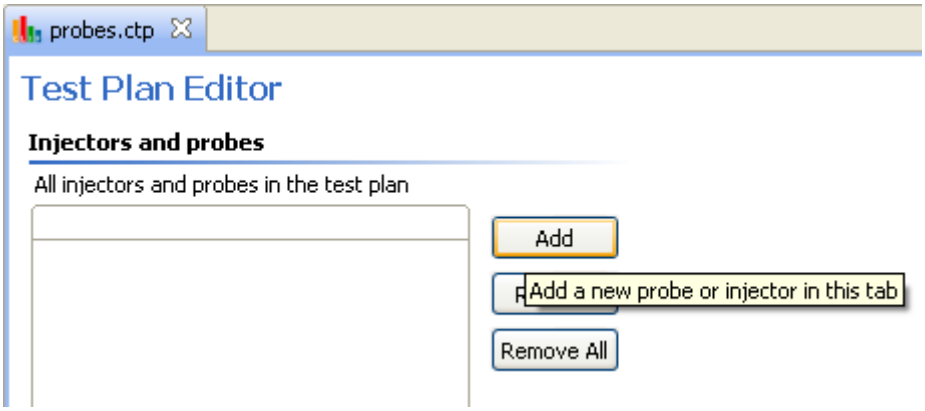
Enter `probes.ctp` as file name and click on "Finish".

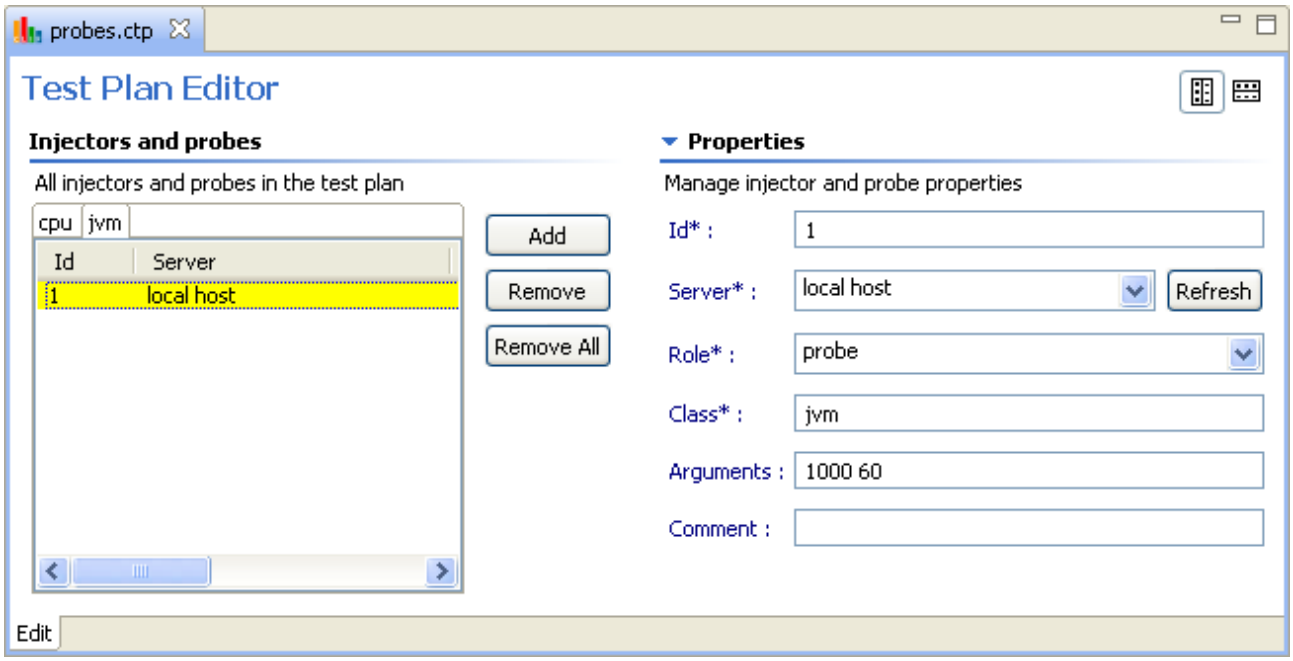


2.3 Add a JVM probe

Let's create a probe to monitor the JVM's memory usage. Click on the "Add" button and enter the following information:

- leave the proposed Id (you may also change it for whatever value, as long as it is meaningful to you, and unique among all injectors and probes in this test plan);
- keep `local host` as Server (the probe will be deployed in the console itself);
- ensure the `probe` Role is selected;
- enter `jvm` in the Class field (caution: this field is case sensitive);
- enter arguments `1000 60`. The first number indicates the polling period in milliseconds (then a memory usage measure will be performed each second), while the second argument sets the monitoring duration in seconds (the probe will be working for one minute, and then will stop).



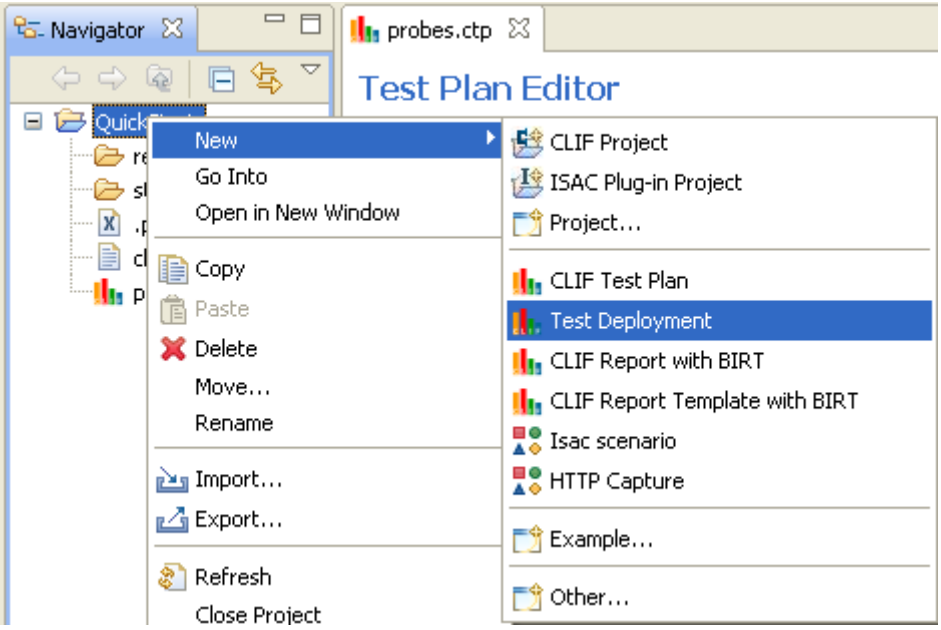


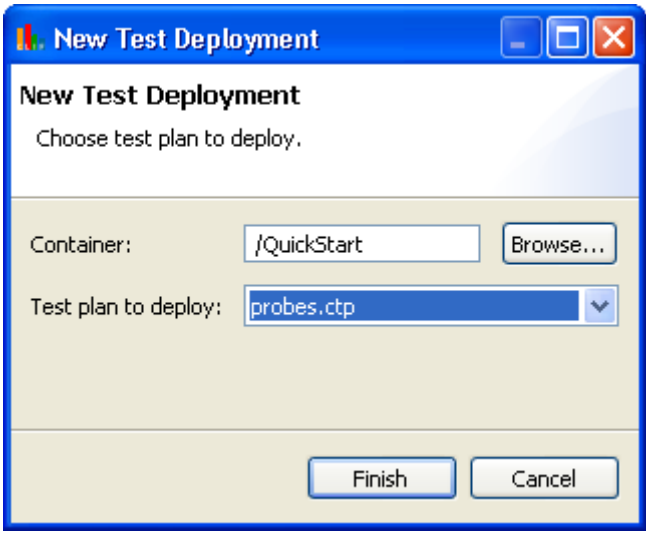
Refer to the User Manual to get the list of available probes. Unlike the JVM probe, most of them are operating system-dependent.

Now, your test plan is ready. You need to deploy it in order to execute it.

2.4 Test Deployment

Deploy it with wizard "New → Test Deployment" and validate with the default parameters.

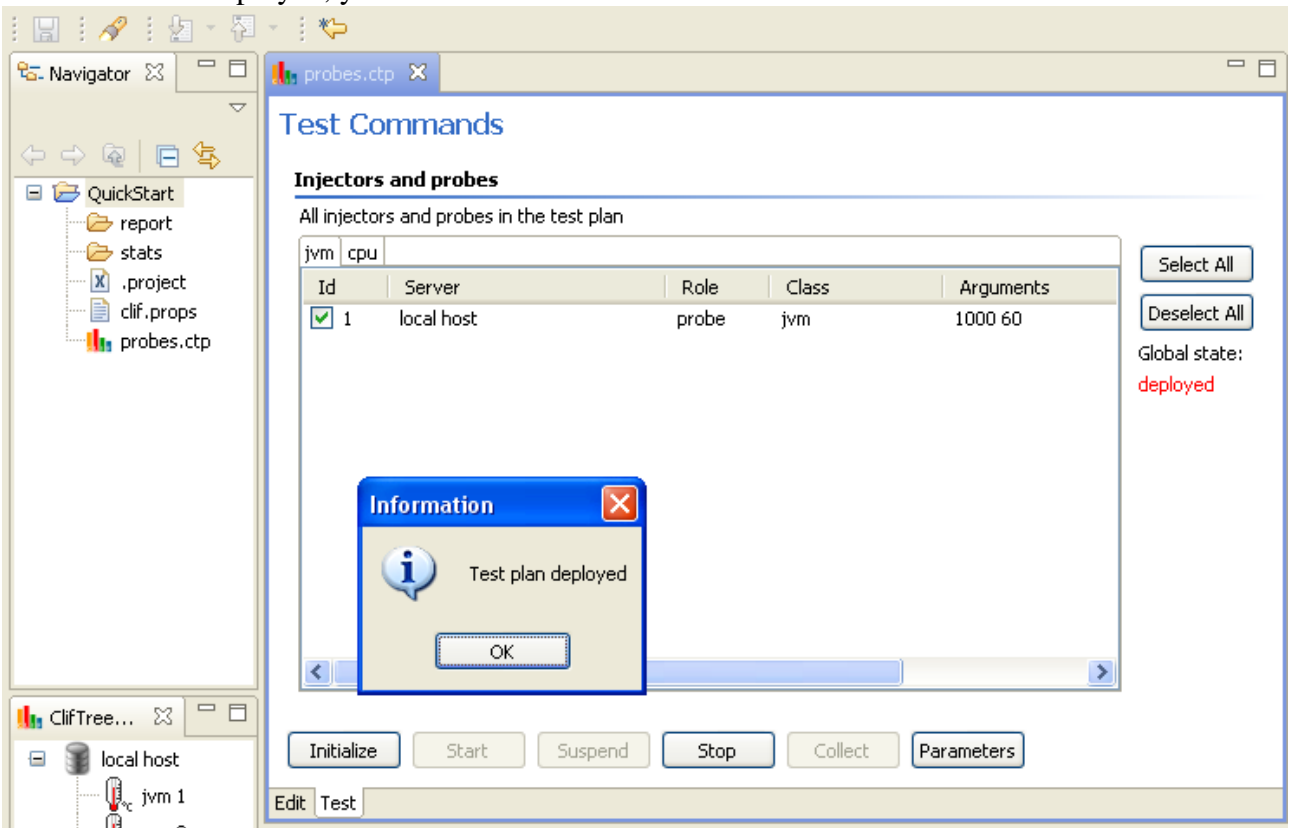




Click on button "Finish" to launch the test plan deployment.

2.5 Test execution

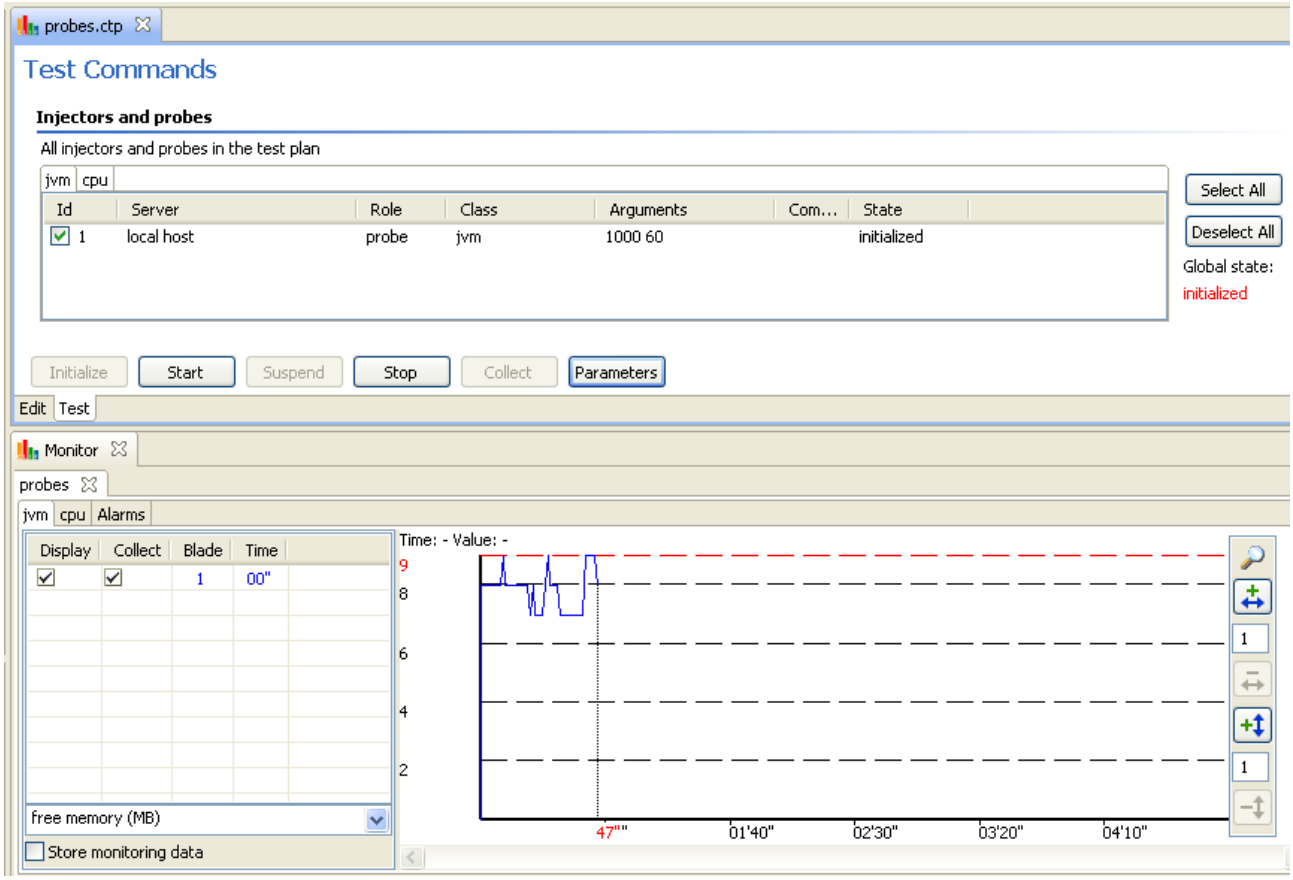
Once the test is deployed, you must initialize the test.



2.5.1 Initialize

The initialization step bootstraps probes and injectors but does not start the test.

Initialize the test, keeping the default given test id.

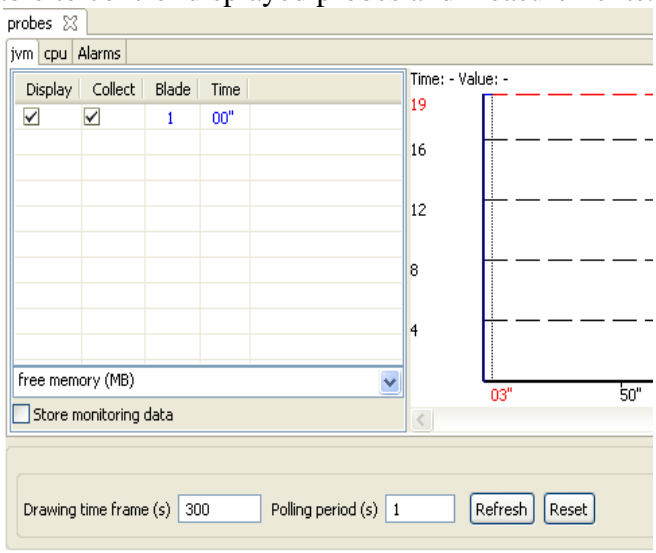


The JVM probe starts monitoring memory usage, but its execution duration is counted only when it is running (i.e. started and not suspended).

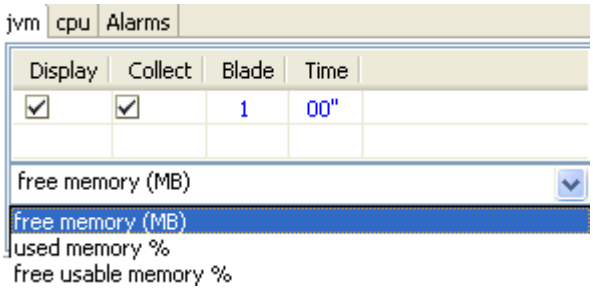
2.5.2 Monitor tabs

The monitor view allows you to see the details of each type of probe or injector. An extra tab shows all alarm events sent by probes.

In each tab, you will be able to control displayed probes and measurements.

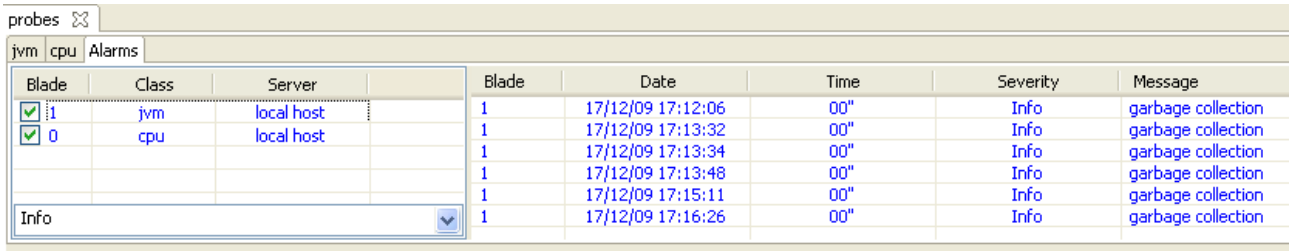


jvm probe



The JVM probe monitors free memory in currently allocated heap (MB), used memory % with regard to currently allocated heap, free % of maximum allocatable memory heap.

Alarms



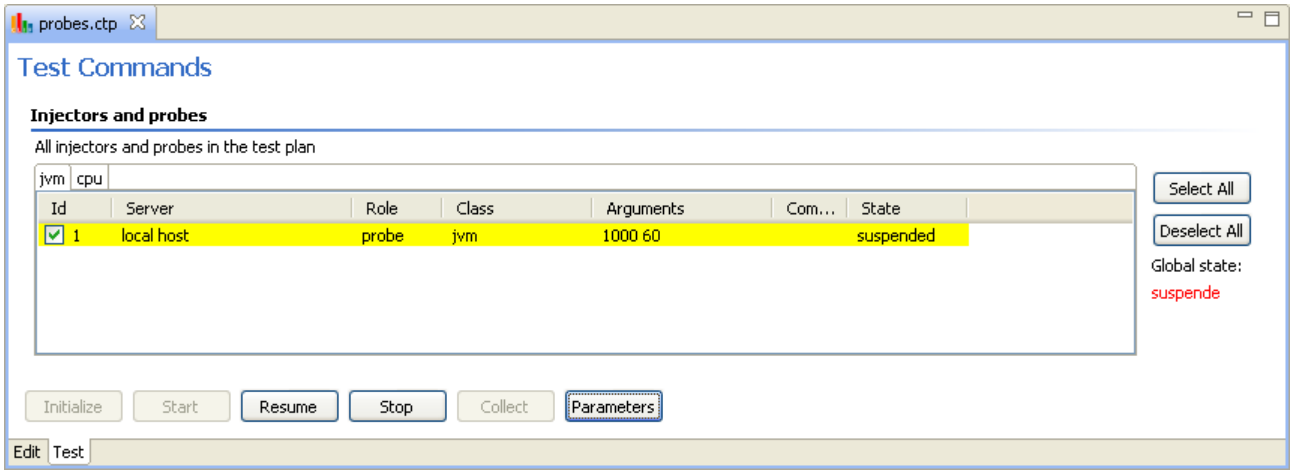
Alarm messages appear depending on some events occurring in some probes or injectors. For example, the JVM probe raises information-level alarms when garbage collection occurs.

2.5.3 Start

Starting the test will begin the test and store data (mainly measurements). From the start, the test will last 60 seconds as defined in the JVM probe.

2.5.4 Suspend/Resume

You can suspend the test during execution and resume it when desired. No data is stored and no load injection is performed during suspension time. Probes go on producing measurements, but their execution time is frozen.



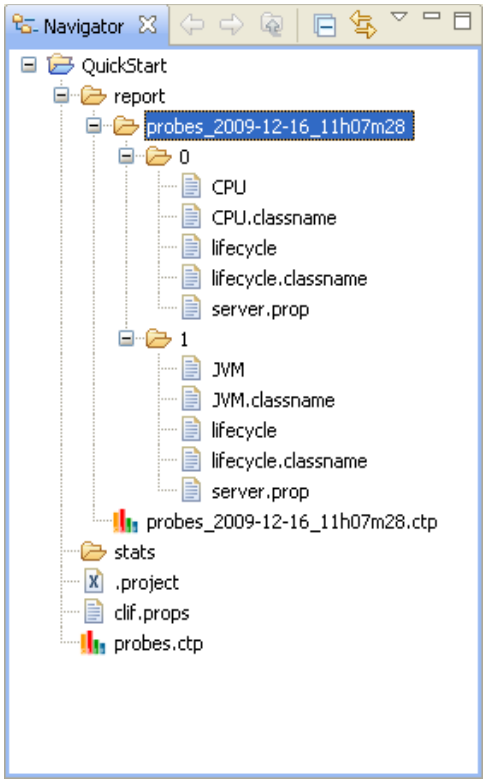
Once the execution time has elapsed, the test automatically stops.

2.5.5 Collect

Once the test is completed, you may collect data in the report directory with the "Collect" button. This should be immediate since the probe is local.

2.6 Results

Once collected, test data are available in the `report` directory.



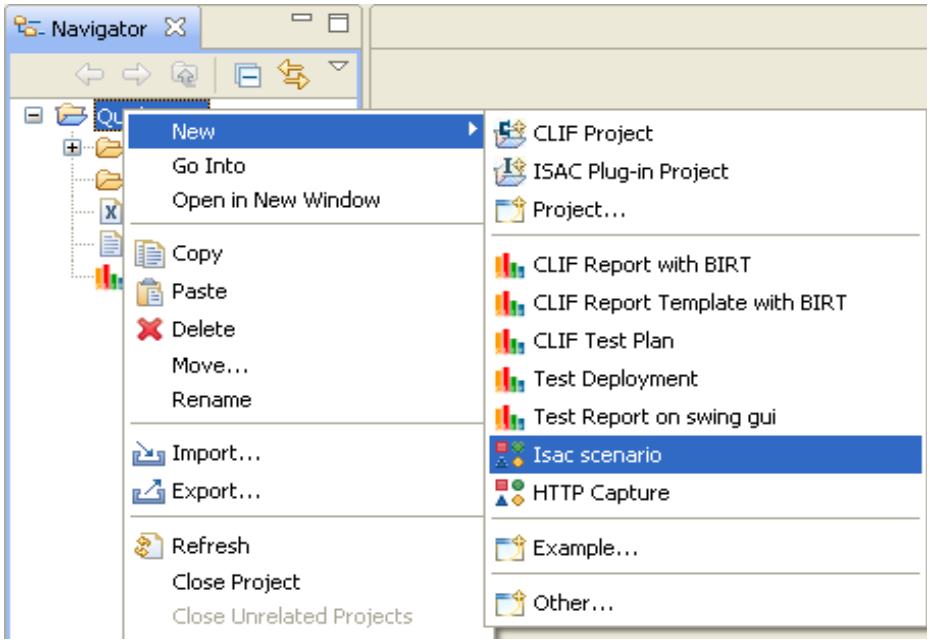
3 A scenario for web testing

Let's create a scenario to generate HTTP Get requests. This scenario defines a sequence of actions and *think times* to be executed during the test; such a sequence is called a *behavior*. This behavior will be replicated and executed in parallel by *virtual users*. The number of simultaneously active virtual users running this behavior, according to elapsed test time, is specified by a *load profile*.

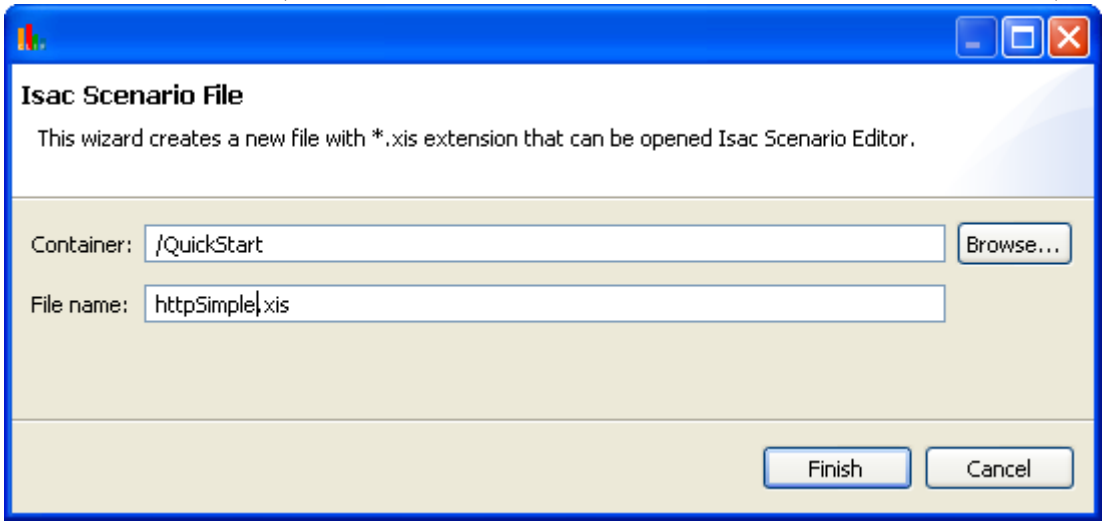
As for almost every utilization of CLIF, we use CLIF's ISAC extension, dedicated to scenario definition and execution.

3.1 Create a scenario with the ISAC environment

First, let's create an Isac Scenario with the "New → Isac Scenario" wizard



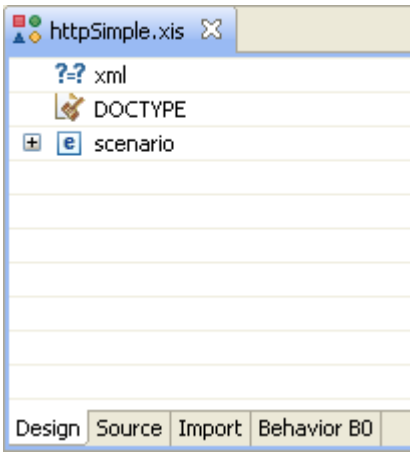
Enter the scenario file name (the .xis file name extension stands for "XML Isac Scenario").

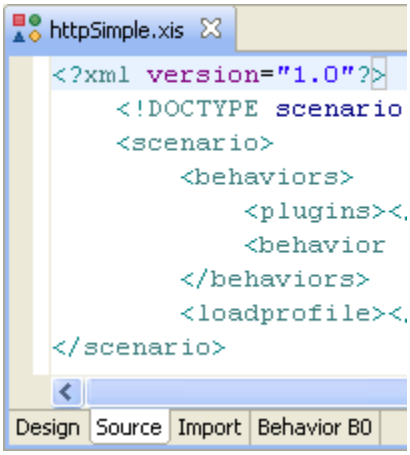


3.2 Discover the ISAC scenario editor

3.2.1 Isac Editor's tabs

At the bottom part of the scenario editor, you can see 4 tabs, namely Design, Source, Import and Behavior B0. An Isac scenario is stored as an XML file. The Design tab shows a tree view of the XML content, while the Source tab is a generic XML editor. Using the XML editor can be convenient in some cases, but risky: use with care if you want to keep your scenario meaningful to ISAC!





The Import tab lets you import all ISAC plug-ins that are necessary to your scenario: support for injection protocols (HTTP, FTP, SIP, etc.), data set generation, or think times.

Finally, the Behavior tab provides a high level editor for defining a virtual user behavior. It is the recommended way to edit behaviors, since it is much more user-friendly and safe than the Source

edition tab. This behavior's default name is "B0". You may define as many behaviors as you like, all identified by whatever unique name.

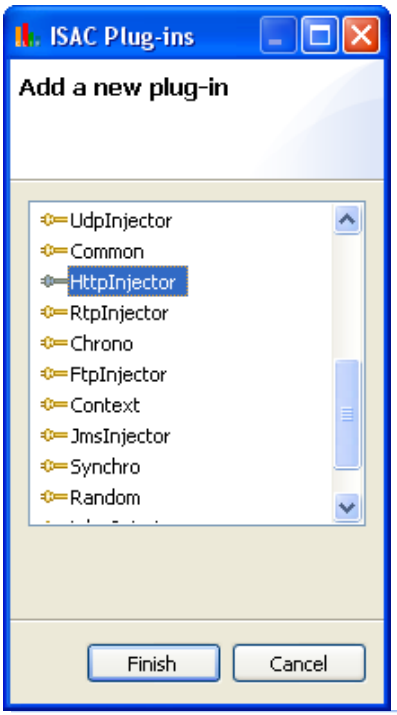
3.3 Writing a scenario

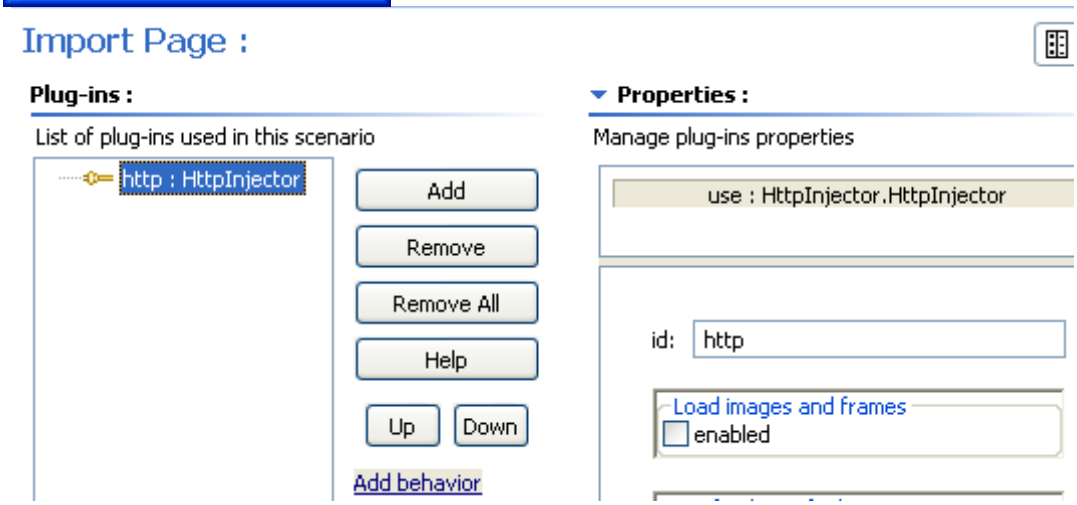
3.3.1 Import necessary plug-ins

Since we want to make some web testing, we need to import the HttpInjector plug-in, which provides all the client primitives of the HTTP protocol.

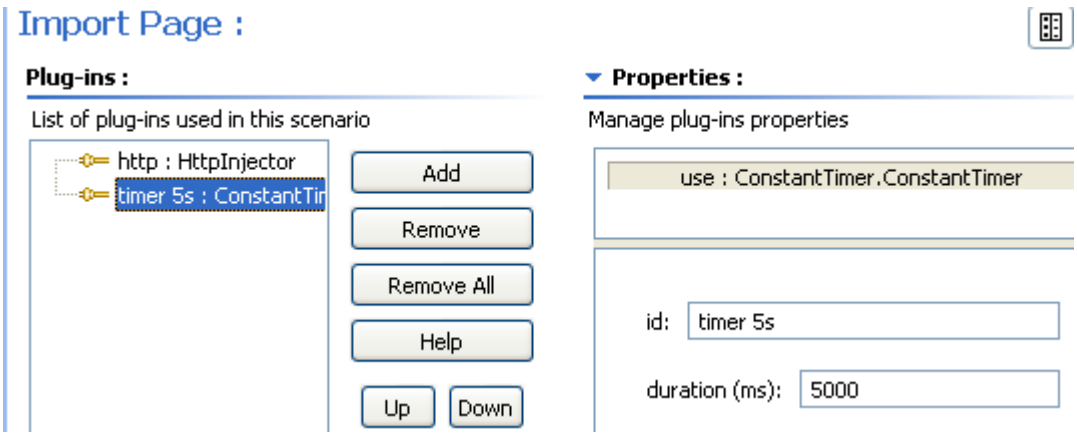
Go back to the Import tab, click on the "Add" button, and select HttpInjector in the plug-in list.

It is recommended to rename the imported plug-in's id from the default name (HttpInjector_0) to a simpler name, say http. Save the scenario.





Then, we also need a plug-in to produce think times. The ConstantTimer plug-in provides support for constant think times. Rename its import name to "timer 5s", and set the timer duration to 5000 ms. Save the scenario.

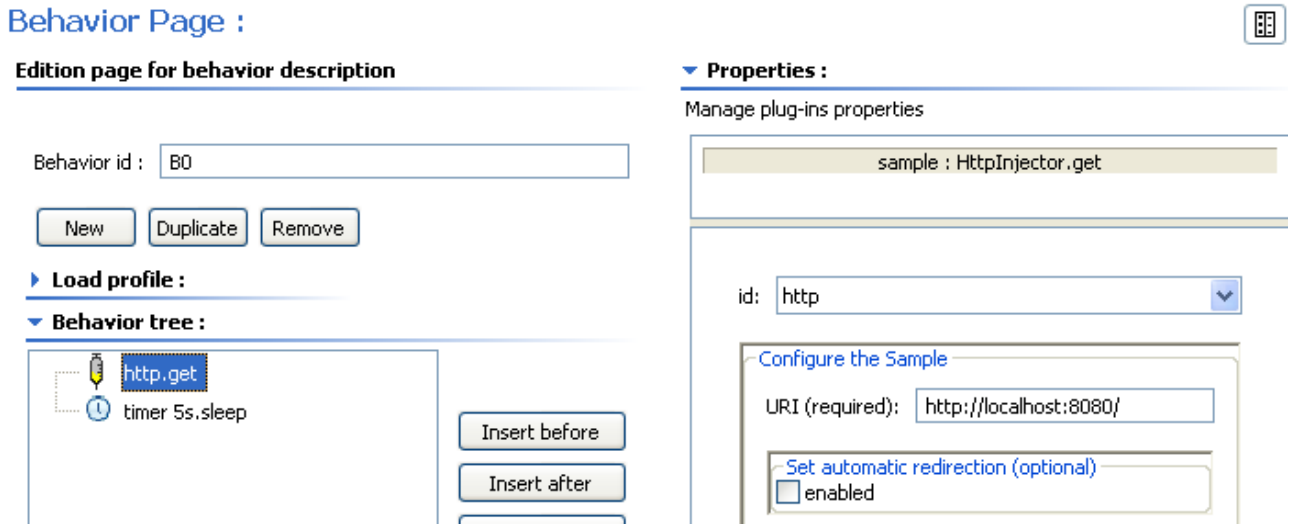


Now, we have all necessary plug-ins for our test.

3.3.2 Write a virtual user behavior

Switch to the Behavior tab to edit the virtual user behavior.

Insert an `http.get` request (aka *action* or *sample*) and a sleep *timer* from "timer 5s".



This behavior performs an HTTP Get request, sleeps for 5 seconds, and finally completes.

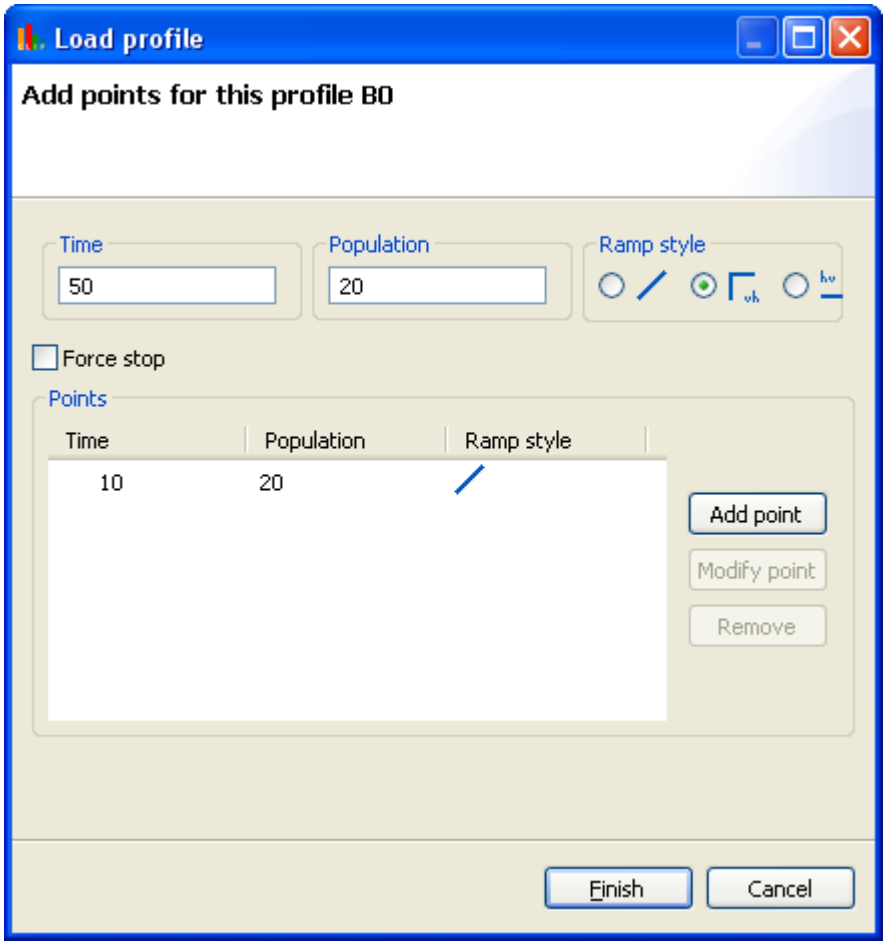
To set the URI to get, click on the http.get request in the Behavior tree, and fill the URI field. Do not enter a public web site. Use your own web site instead, in order to avoid flooding a public web site with requests. For example, you may launch a light Servlet container like Jetty (http://www.eclipse.org/jetty/downloads.php). You just need to get the distribution, unzip it and launch it with command `java -jar start.jar DEBUG=true`. The server will be accessible at http://localhost:8080/ and you will have direct feedback of the requests with debug logs.

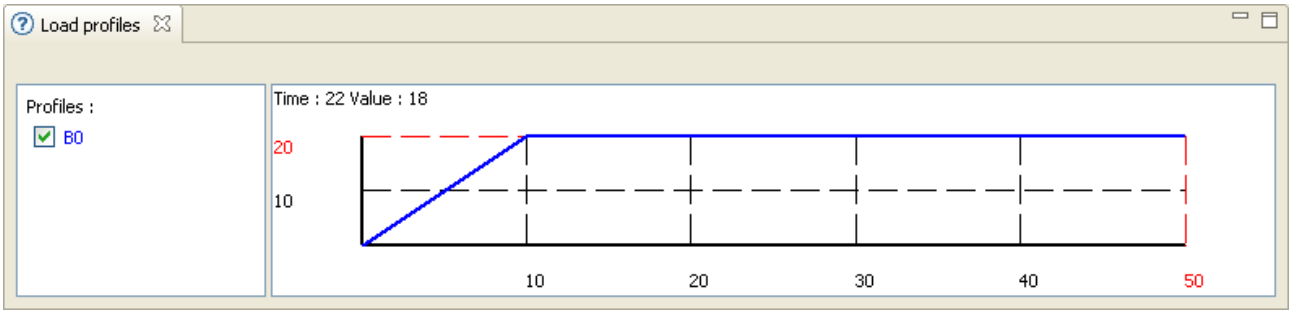
3.3.3 Load profile

Still in the Behavior tab, click on button "Create" in the load profile section. The load profile editor consists of a list of points (*execution time in seconds*, *number of active virtual users*). The transition between one point to the next one may be linear or square, with 2 possible squares: vertical and then horizontal, or horizontal and then vertical. When not specified, the initial point at time zero is implicitly set to (0,0).

Enter points (10, 20) and (50, 20) with a linear ramp style.

Click on button Finish, save, and view the result on the Load profile view: the test starts with 0 virtual users. Then, there is a ramp-up to 20 virtual users at 10s execution time, and then a plateau with 20 virtual users until 50s execution time.





Your scenario is now ready to use.

3.4 Make and run a test plan with a load injector

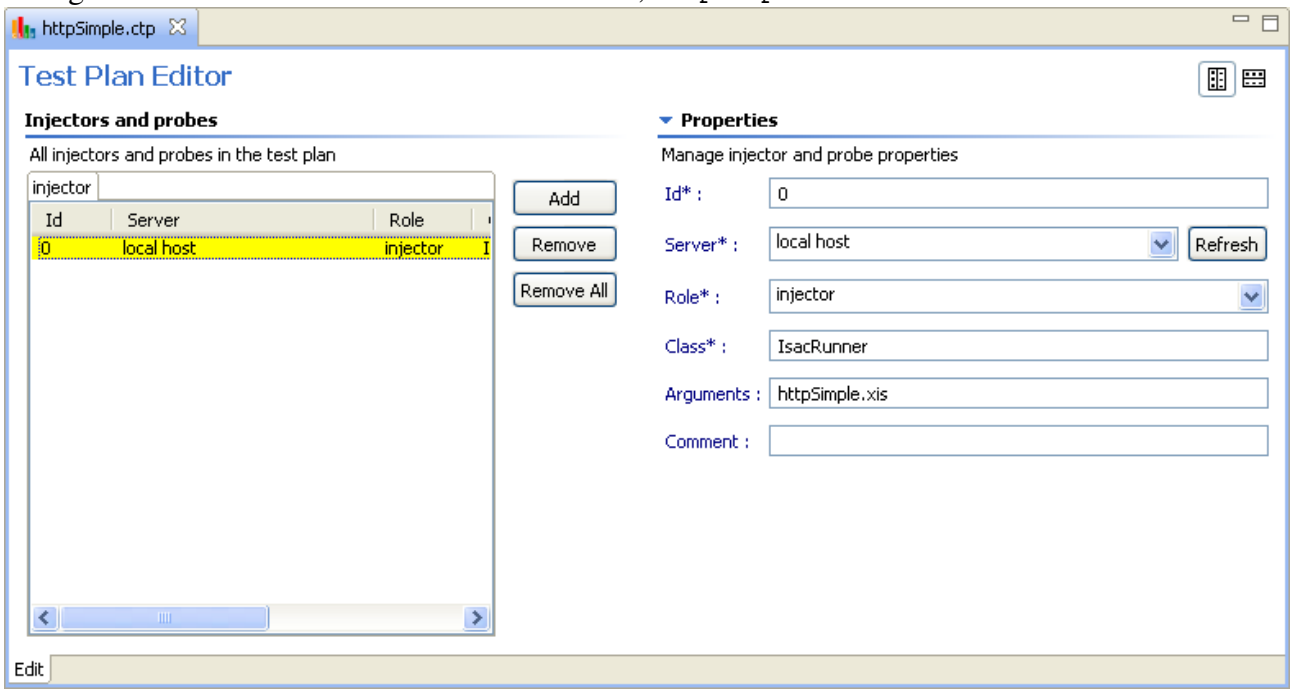
3.4.1 Create a test plan

To deploy and run this scenario, we must define a CLIF test plan including a load injector running the scenario. First, create a test plan named `httpSimple.ctp` (see section 2.2).

3.4.2 Add a load injector to the test plan

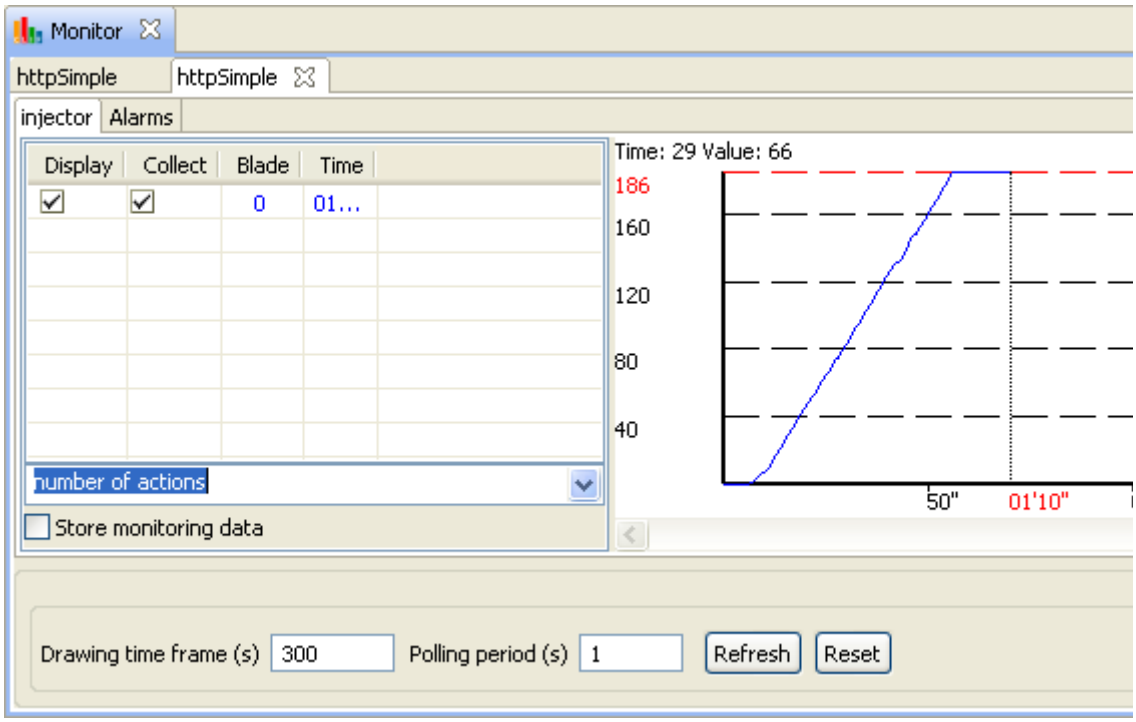
Click on the "Add" button and set the following properties:

- Id: keep the default value or set any name meaningful to you;
- Server: keep `local host`;
- Role: select `Injector`;
- Class: enter `IsacRunner`, which specifies that the load test will use the ISAC extension (caution: this name is case-sensitive!);
- Arguments: the name of the ISAC scenario file, `httpSimple.xis`



3.4.3 Deploy and run

Deploy the test plan (see section 2.4). Then, initialize and start the test (see section 2.5). During test execution, you may monitor some monitoring statistics about response times, throughput and errors.



When the test execution is complete, you may collect measurements in order to analyze them later (see section 2.5.5). Furthermore, a quick statistical synthesis on response times is automatically generated in the `stats` directory of the CLIF test project (look for file `quickstats.csv`).

4 Http capture and replay

4.1 Rationale

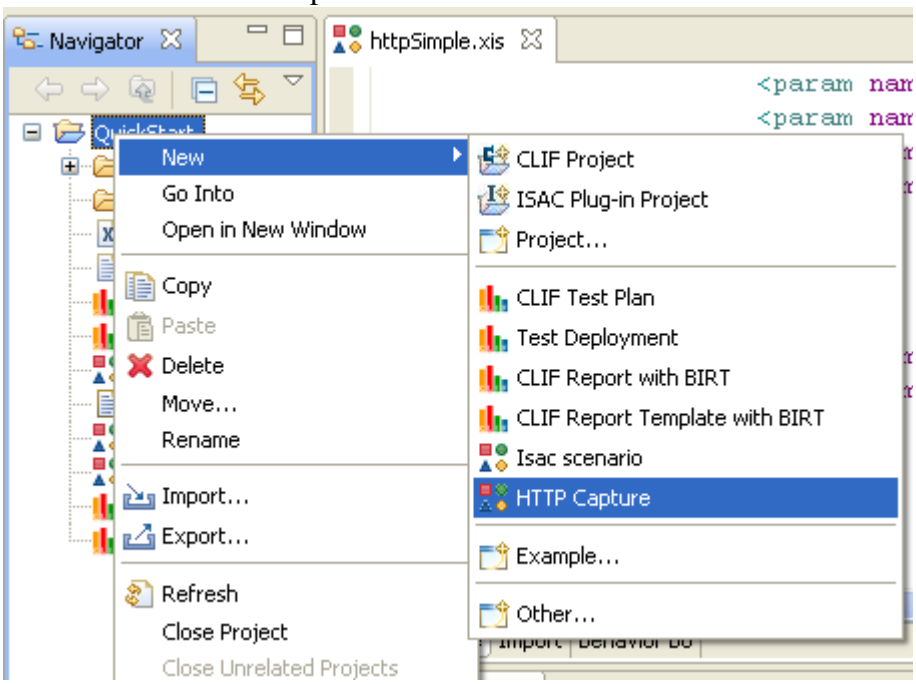
When writing a virtual user behavior, it is not always easy to know all the requests that will be issued. A convenient way of defining a behavior for web testing is to enhance a web browser with some special tooling that can record all HTTP requests, and possibly the real think times, and generate a virtual user behavior.

The CLIF console provides such a tool, able to generate an ISAC scenario from a real user session. Basically, it is an HTTP proxy that records URIs and measure think times.

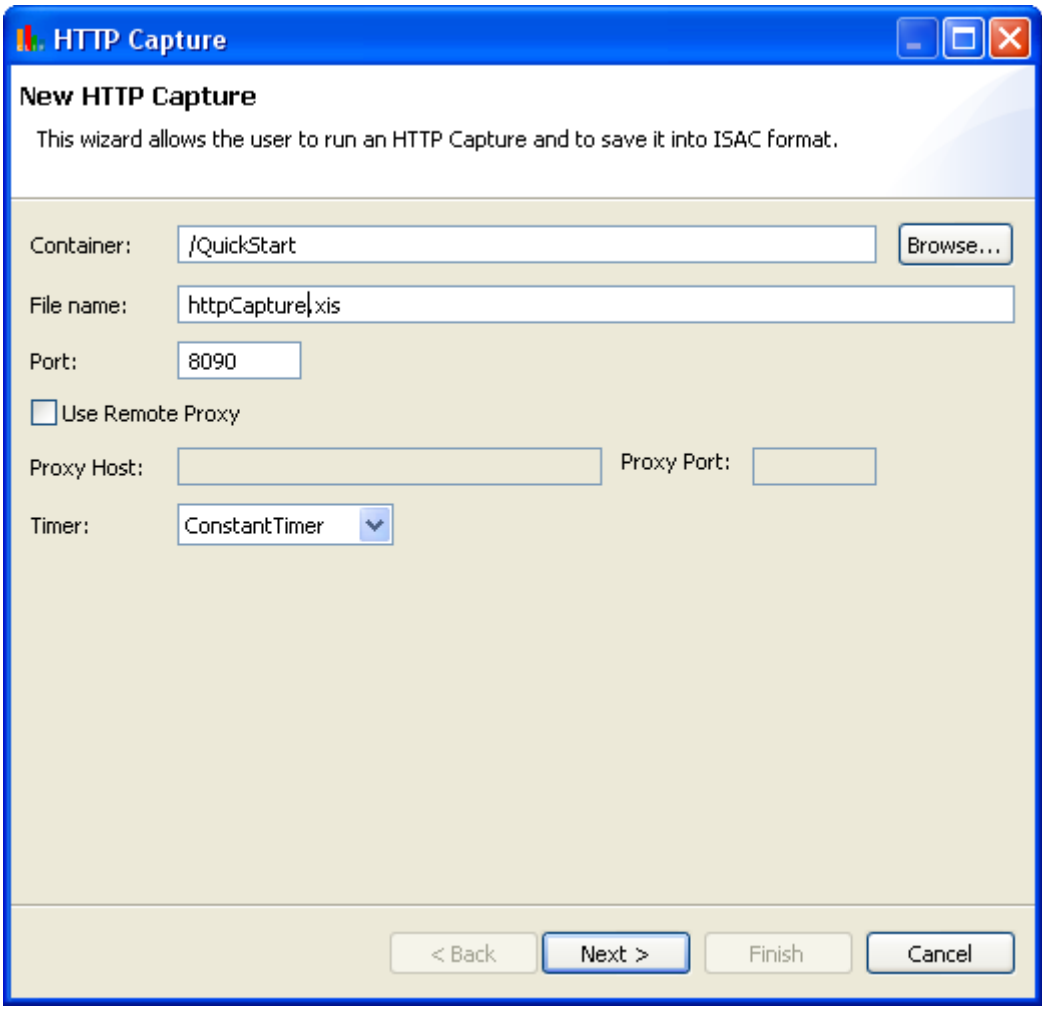
4.2 Capturing a real web user session

4.2.1 Launch Http capture

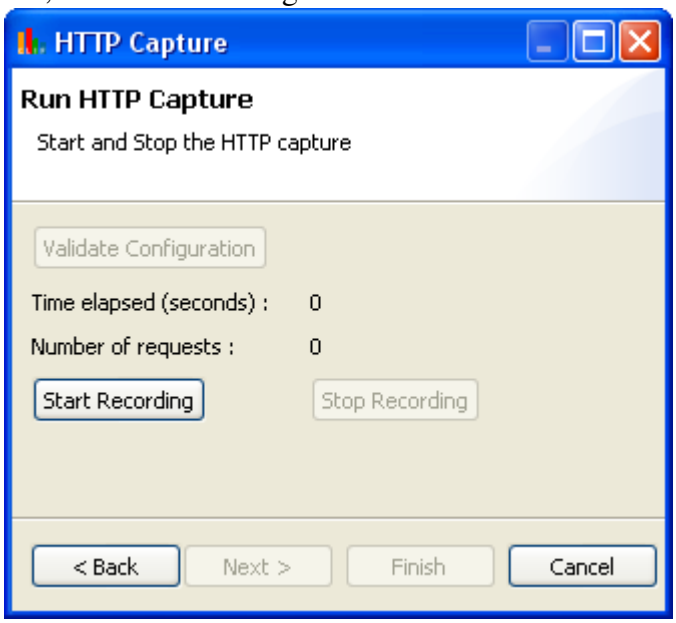
First, select wizard "New → HTTP Capture"



Name the capture file `httpCapture.xis` and click on button "Next".



On the next dialog window, validate the configuration to start the embedded HTTP proxy.



Now, you just have to click on button "Start Recording" to make the proxy record URIs and think times.

4.2.2 Configure your navigator to use the capture proxy

Firefox

To use the proxy in Firefox, go to Tools → Advanced → Network tab → Parameters

Enter the proxy parameters (localhost, port 8090) and validate.

Then, just use Firefox as usual to get visited URIs recorded by the proxy.

Internet explorer

Right-click on Internet Explorer and chose Properties

Go to the connection tab and chose network parameters

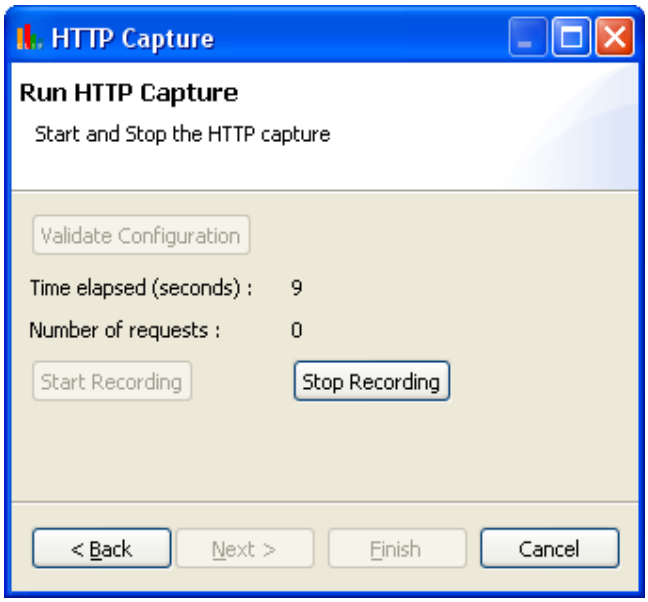
Enter the proxy parameter (localhost, port: 8090) and validate

Note

Don't forget to remove this proxy configuration once you are done with your capture!

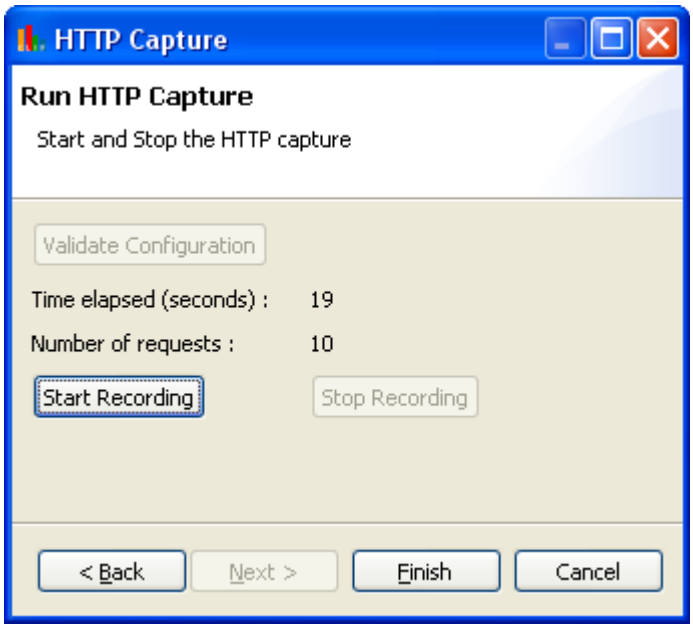
4.2.3 Record the scenario

Start the recording.



Then, just use your browser as usual to get visited URIs recorded by the proxy. The number of requests should increase accordingly.

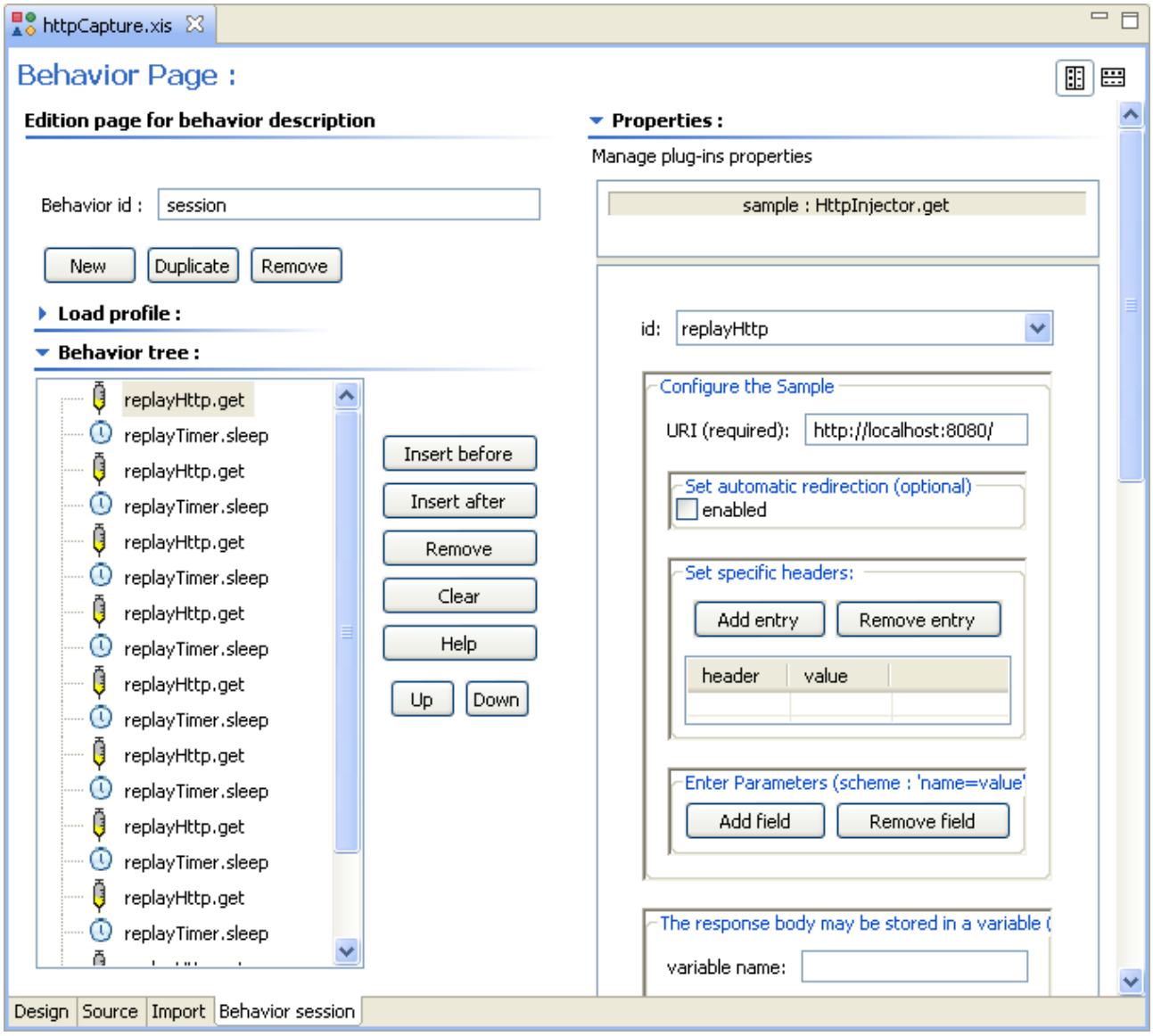
Stop recording when you are done, and click on "Finish".



4.3 Edit scenario httpCapture.xis

The wizard has generated scenario file in your CLIF test project (see the project navigator view). Open this file (`httpCapture.xis`).

You need to define a load profile before using this scenario in an injector (see 3.3.3). You might as well edit the behavior to match your needs (change some parameters, add or delete requests or think times, etc.).



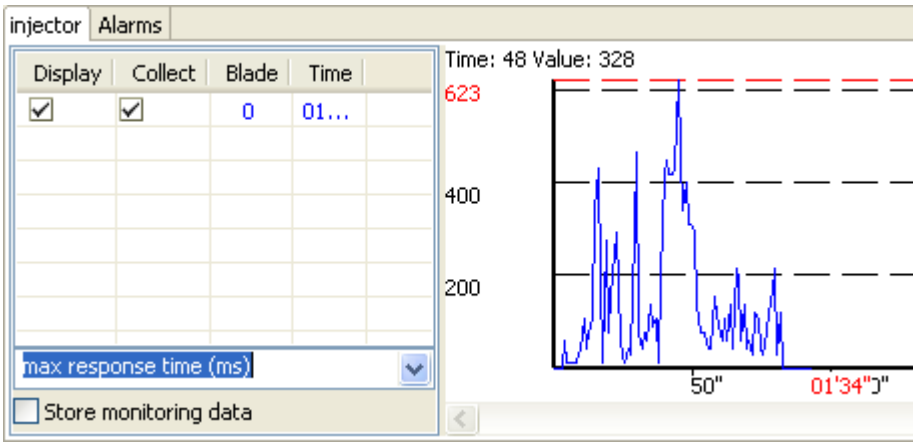
4.4 Deployment and execution of httpCapture.xis

Create a new test plan `httpCapture.ctp` (see 2.2) and add an injector (see 3.4.2) with the following parameters:

- Role: `Injector`
- Class: `IsacRunner`
- Arguments: `httpCapture.xis`

Deploy the test plan (see 3.4.3)

Initialize, start and collect results.



You have completed the Quick Start guide. Now, you may discover other CLIF features by reading the User Manual or having a look at the sample scenarios and test plans: other injection protocols, external data sets, ISAC language, other user interfaces, etc.

Java developers may also extend CLIF with new probes and injectors (see the Developer Manual).

Enjoy!