February, 11th 2010

CLIFv2 installation guide

http://clif.ow2.org/

Table of contents

1. How to get CLIF working?

1.1. Technical requirements

CLIF framework and provided load injectors are 100% Java™. CLIF requires a Java development kit[1] (JDK), and the Java-based ant utility from Apache.org. Current CLIF version is known to be working with:

- Sun JDK 5.0 (also known as J2SDK™ 1.5) or Sun JDK 6.0
  Download from http://java.sun.com/javase/downloads/index.jsp
- Apache ant utility version 1.5.4 or greater
  download from http://ant.apache.org/bindownload.cgi
  Make sure ant is using the right JDK!
- Linux 2.4 and 2.6 kernels
- MacOS.X tiger
- Microsoft Windows XP™

System probes for Linux are also 100% Java, while system probes for Windows and MacOS.X are native (C-code embedded in Java code via the Java Native Interface).

Since CLIF is written in Java, the only constraint about the SUT is that it must be reachable from a Java Virtual Machine (JVM), either directly or indirectly through some wrapping, gateway or native library.

There are two ways of getting a CLIF runtime environment: either by getting the whole source from the Subversion repository, or by getting a ready-to-use binary distribution.

1.2. Ready-to-use distributions

CLIF's site at ObjectWeb Forge offers several binary distributions, available as zip files (see http://forge.ow2.org/project/showfiles.php?group_id=57):

- `clif-2.0.0-eclipseconsole-linux.zip`
  Eclipse RCP based standalone console for Linux/Intel
- `clif-2.0.0-eclipseconsole-windows.zip`
  Eclipse RCP based standalone console for Windows/Intel
- `clif-2.0.0-eclipseconsole-macosx.zip`
  Eclipse RCP based standalone console for Mac OS X/Intel
- `clif-2.0.0-eclipseplugins-linux.zip`
  CLIF plugins distribution for Eclipse/Linux. It allows to use CLIF from your Eclipse environment.
- `clif-2.0.0-eclipseplugins-windows.zip`
  CLIF plugins distribution for Eclipse/Windows. It allows to use CLIF from your Eclipse environment.
- `clif-2.0.0-eclipseplugins-macosx.zip`
  CLIF plugins distribution for Eclipse/Mac OS X. It allows to use CLIF from your Eclipse environment.

1 Basically, CLIF could work with a JRE, except that a JRE does not provide a "server" JVM. JDKs provide such a JVM which is more suitable for CLIF use.

- `isac-DnsInjector.zip`
  Support for DNS traffic injection within ISAC scenarios.
- `isac-FtpInjector.zip`
  Support for FTP traffic injection within ISAC scenarios.
- `isac-HttpInjector.zip`
  Support for HTTP(S) traffic injection within ISAC scenarios.
- `isac-JdbcInjector.zip`
  Support for traffic injection on databases using a JDBC driver, within ISAC scenarios.
- `isac-JmsInjector.zip`
  Support for JMS traffic injection within ISAC scenarios.
- `isac-LdapInjector.zip`
  Support for LDAP traffic injection within ISAC scenarios.
- `isac-SipInjector.zip`
  Support for SIP traffic injection within ISAC scenarios.
- `isac-SocketInjector.zip`
  Support for TCP traffic injection within ISAC scenarios.
- `isac-UdpInjector.zip`
  Support for UDP traffic injection within ISAC scenarios.
- `isac-RtpInjector.zip`
  Support for RTP traffic injection within ISAC scenarios.

You may unzip these files wherever you like, except Eclipse plug-ins that you may install in your Eclipse environment (see section 2.3) and isac-xxxInjector that you may install in your Eclipse RCP environment or Eclipse RCP standalone console. Avoid installing a distribution in a directory containing a whitespace character in its path (set-up problems have been reported in some conditions).

1.3. Configuring CLIF

From now on, you are supposed to set CLIF's runtime environment root directory as your current directory. You may configure CLIF either by editing file `clif.props` in `etc/` subdirectory, or by using command "`ant config`". In the latter case, the following questions will be asked:

- *please enter the registry host:*
  enter the IP address or name of the computer where you will run the Registry, either embedded in the Swing or Eclipse console GUI, or launched by command line.
- *please enter the registry port number:*
  enter the port number configured for the Registry, either embedded in the Swing or Eclipse console GUI, or launched by command line. Usual value is 1234.
- *please enter the code server host:*
  enter the IP address or name of the computer where the code server will run, either embedded in the Swing or Eclipse console GUI, or launched by the deploy batch command.
- *please enter the code server port number:*
  enter the port number configured for the code server, either embedded in the Swing or Eclipse console GUI, or launched by the deploy command line. Usual value is 1357.

This configuration operation must be done everywhere you want to run a CLIF server or a console. You may also make this configuration step only once, and copy the resulting file `etc/clif.props` wherever needed.

Note that this configuration utility uses file `etc/clif.props.template` as a template. You may edit this file to change some default Java properties so that any further configuration will keep your changes.

Should you edit file `etc/clif.props`, refer to the appendix on System properties page 15.

1.4. Checking Clif version and execution environment

Use command "`ant version`" to get the version numbers of Java environment, operating system and CLIF. Command "`ant -version`" gives the ant version.

1.5. Upgrading from CLIF v1.x.x

As of version 2.0 of CLIF, some changes have occurred that introduces incompatibility issues with previous 1.x.x versions. The main change is about renaming all reference to org.objectweb to org.ow2.

These incompatibilities affect:

- CLIF test plan files (.ctp) with regard to the probes' fully-qualified class names
- ISAC scenarios, with regard to
  - the renaming of some ISAC plug-ins for the sake of homogeneity
  - the change of XML DOCTYPE declaration
- ISAC plug-ins, with regard to
  - Java source files, because of the renaming of package org.objectweb into org.ow2
  - XML descriptors, because of a change of XML DOCTYPE declaration
- measures collected into the `report` directory, with regard to
  - the copied test plans (see above)
  - the events `.classname` files
- BIRT reports.

To easily cope with this, a translation tool is provided via the `ant` utility and the `build.xml` file provided in the `dist` module (see section Developer Manual):

- `ant -f /path/to/dist/build.xml 2clif2`
  First, this command recursively copies the content of current directory to a new directory named 2clif2-output. Then, it recursively looks for `.java`, `.xis`, `.ctp`, `.xml`, `.classname` and `.rptdesign` files in this copy directory, and proceeds with all necessary translations.

2. Installation

2.1. Introduction

All parts are available as separate Eclipse plug-ins, or delivered as a single ready-to-use standalone program. Note that CLIF's runtime environment directory, as often referred to in this documentation, is located in the console plug-in path, i.e. something like `plugins/org.ow2.clif.console.plugin_x.x.x/`

Please refer to the on-line help for detailed documentation about these parts.

As an Eclipse applications, a number a useful options may be set on the command line, such as:

- `-consoleLog` to see messages printed out to your terminal;
- `-vm /path/to/the/jvm` to set the right Java Virtual Machine to be used;
- `-data /path/to/my/workspace` to use a different workspace directory from the default one.

Should you require to set some specific system properties, please edit file `clif.props.template` in directory `plugins/org.ow2.clif.console.plugin_x.x.x/etc`.

2.2. Install the Eclipse-RCP based standalone CLIF distribution

Eclipse RCP based binary distributions of CLIF are full-fledged standalone executables that include Eclipse RCP environment for a specific operating system. They require a JDK, and most often Apache ant utility to run CLIF servers.

You just have to get the right binary distribution for your operating system, or generate it from the source (see Developer Manual) and unzip it wherever you want on your computer, avoiding path names with whitespace characters.

Finally, run clif-console program with any useful argument (as detailed above).

2.3. Install the Eclipse plug-ins

CLIF provides 4 Eclipse plug-ins, namely the clif.console plug-in, the clif.isac plug-in and two clif.oda plug-ins. The clif.isac Eclipse plug-in contains the Isac scenario editor and the Isac plug-in creation wizard. The clif.oda plug-ins provide BIRT reporting tools with access to measures generated by CLIF. The clif.console Eclipse plug-in contains all the remaining parts of CLIF (the test plan editor, the supervision console).

2.3.1. Dependencies

CLIF Eclipse plug-ins depends on a number of Eclipse plug-ins that may not be present by default in your Eclipse Workbench installation: `Xerces`, `BIRT`, `EMF` and `WST` plug-ins. `BIRT` is optional to run CLIF deployment but is needed to run test analysis based on `BIRT`.

2.3.2. Installation

As of version 2.0, CLIF plug-ins rely on Eclipse 3.3 (also known as Europa). Starting from your Eclipse IDE for Java, you may just copy CLIF plug-ins in the `plugins` directory of your Eclipse installation to make CLIF work.

Take care to set write permissions for the `clif.console` Eclipse plug-in directory. Although this is not really satisfactory, you would not be able to run tests without being granted write permissions. As a smarter alternative, you may set the appropriate property in file etc/clif.props, in the `clif.console` plug-in directory, to set the target directory for generated measures (see 1.1).

2.3.3. Execution

Remember to run Eclipse with the right JVM, as mentioned in section 1.1. Use option `-vm` to make sure Eclipse uses the right JVM. See section 2.1 for the most important options you may set when launching your Eclipse workbench.

2.4. Java Swing-based graphical user interface

To run the CLIF Swing console, the mandatory requirement is the right JDK and preferably the Apache ant utility.

You just have to get the right CLIF distribution (see 1.2), or generate it from the source (see Developer Manual). Then, unzip it wherever you want on your computer, avoiding path names with whitespace characters. To run the console, use command `ant console` in the root directory of the unzipped distribution.

2.5. CLIF server

2.5.1. Requirements

You need to download and unzip the clif-2.0.0-server.zip archive on the platforms on which you'll deploy the CLIF servers.

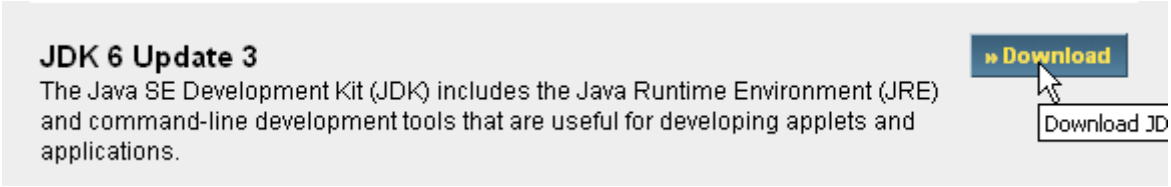
3. Install mandatory requirements

3.1. Download requirements

3.1.1. Sun J2SDK™ 1.5 (1.5 version or greater is mandatory)

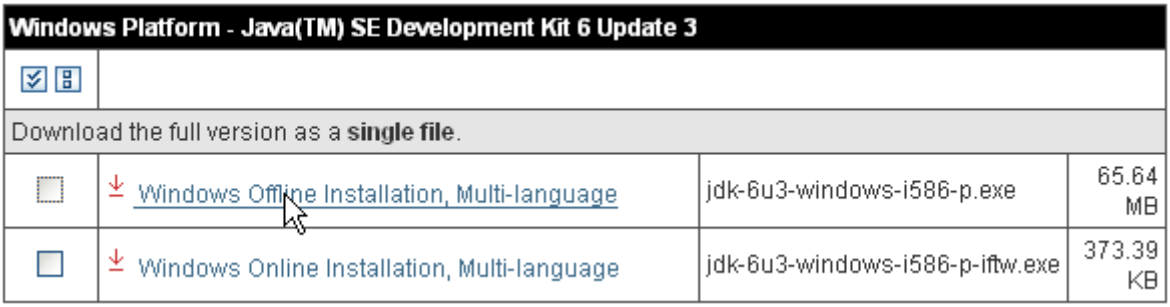
Click on the following link: http://java.sun.com/javase/downloads/?intcmp=1281

Click on the download button of the current version of the JDK:



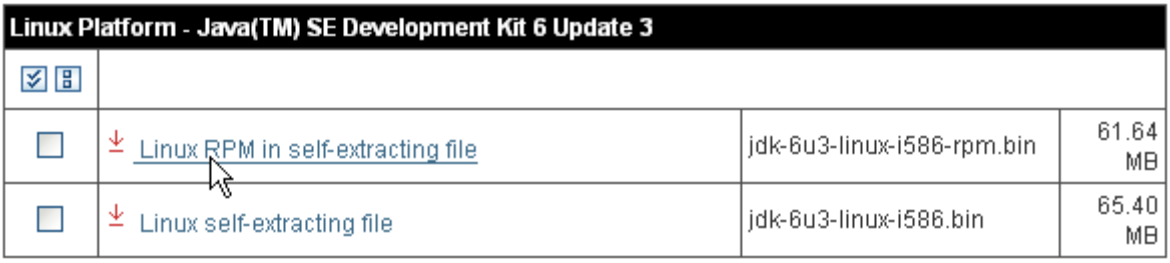
Accept the license agreement:

Select the platform where CLIF will be used:



Windows Platform - Java(TM) SE Development Kit 6 Update 3			
Download the full version as a **single file**.			
	Windows Offline Installation, Multi-language	jdk-6u3-windows-i586-p.exe	65.64 MB
	Windows Online Installation, Multi-language	jdk-6u3-windows-i586-p-iftw.exe	373.39 KB

or



Linux Platform - Java(TM) SE Development Kit 6 Update 3			
	Linux RPM in self-extracting file	jdk-6u3-linux-i586-rpm.bin	61.64 MB
	Linux self-extracting file	jdk-6u3-linux-i586.bin	65.40 MB

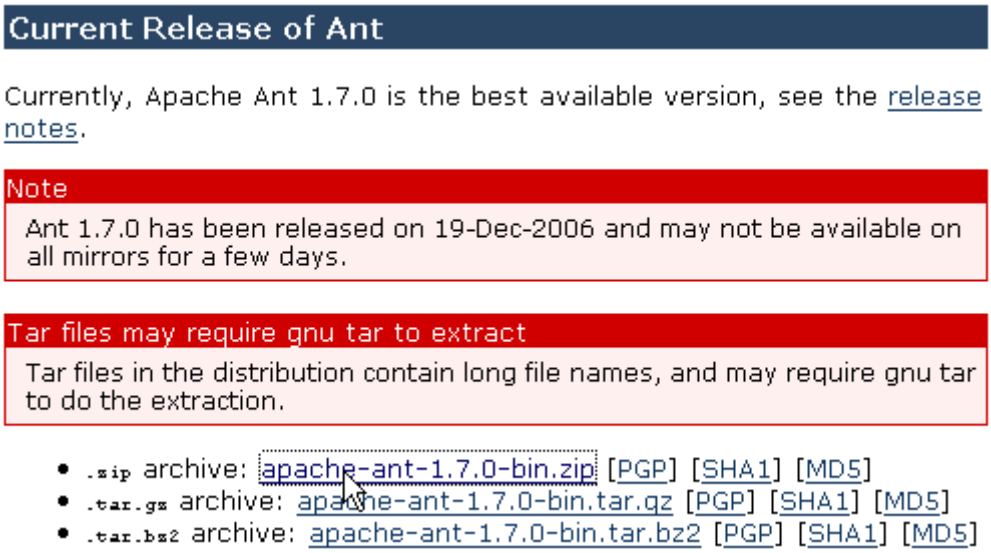
and save the downloading file on your computer.

Once the download done, you can launch the installation on the environment you want to install it.

3.1.2. Apache ant utility version 1.5.4 or greater

Click on the following link: http://ant.apache.org/bindownload.cgi

Click on the current release of apache ant utility depending of the platform where CLIF will be used (For Windows select the .zip file and for Linux the .tar.gz file):



Then save the downloading file on your computer.

Once the download done, you can unzip the archive file where you want on your platform.

3.2. Environment variables setting

3.2.1. Windows OS

Now you have to set the following environment variable:

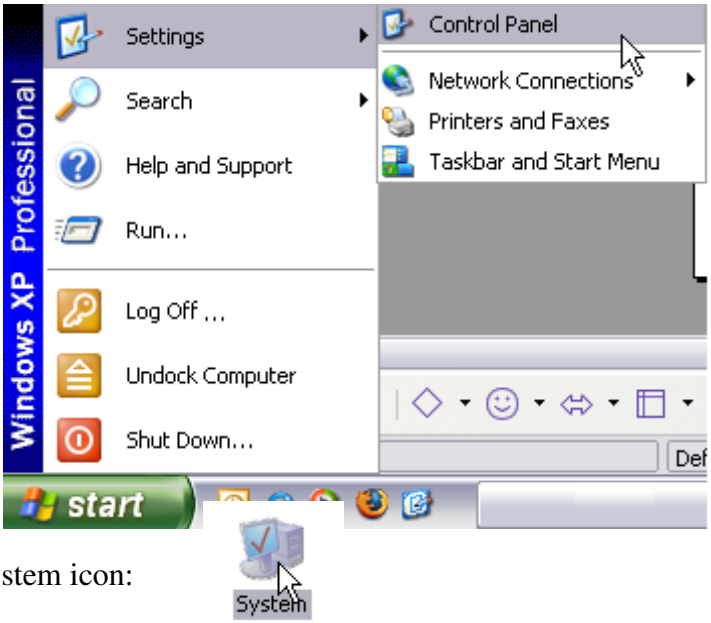
- JAVA_HOME=C:\Program Files\Java\jdk1.6.0_03
- ANT_HOME=C:\Program Files\apache-ant-1.7.0

And to modify:

- PATH=<value already in path>;%JAVA_HOME%\bin;%ANT_HOME%\bin

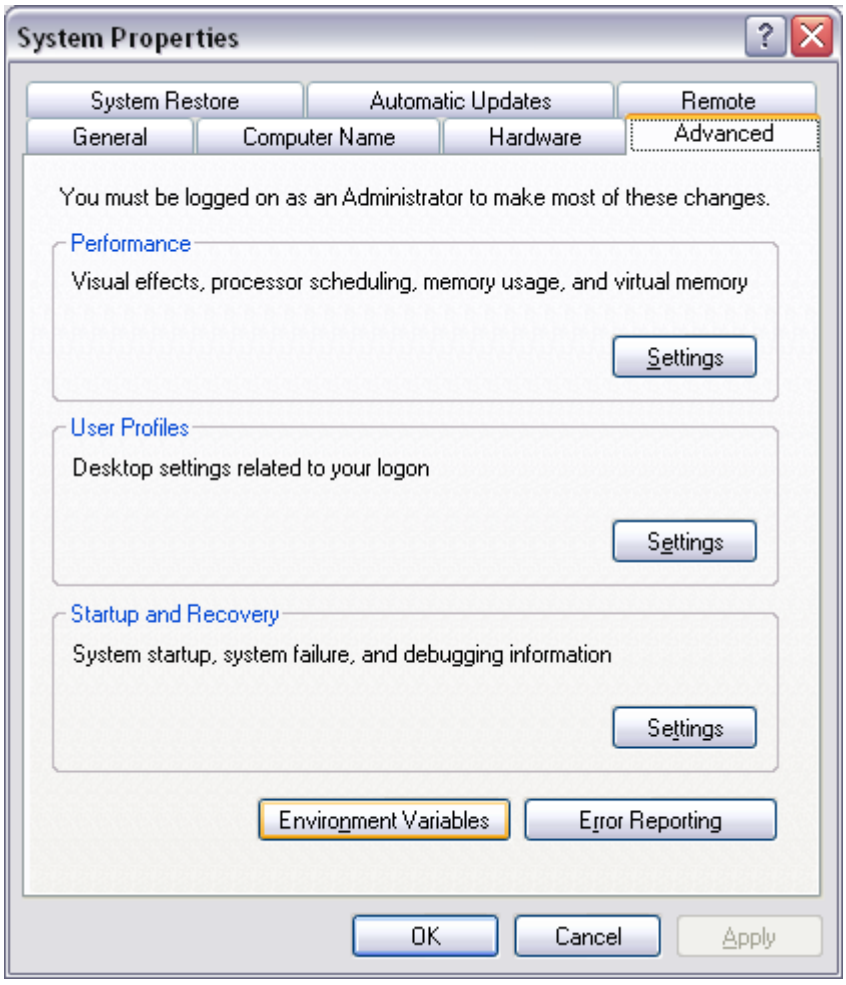
Go to:

Start > Settings > Control Panel

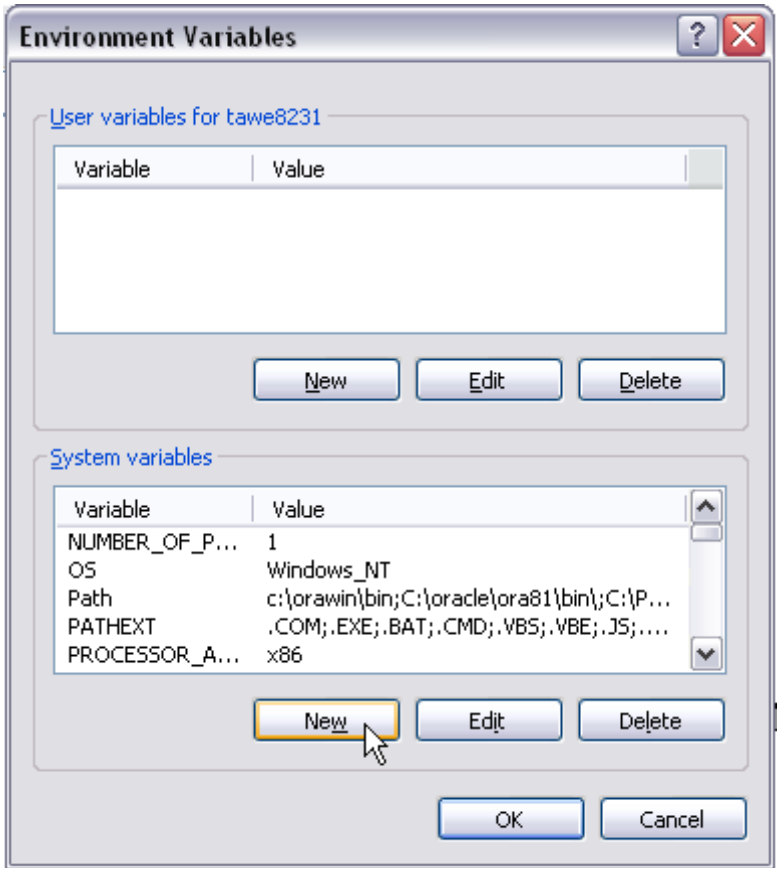


Double click on the system icon:

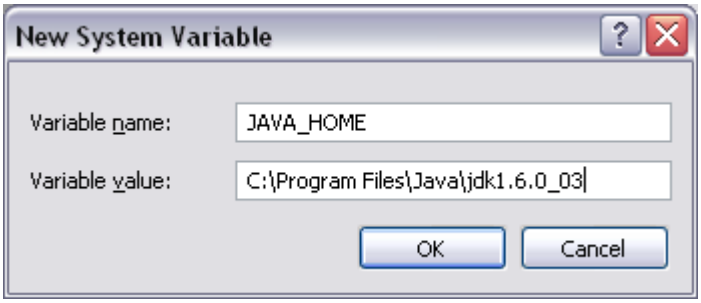
Choose advanced tab and click on the environment Variables button:



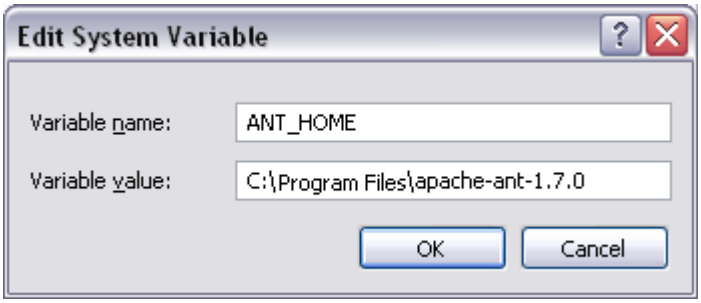
Now click on the New button of the System variables parameters group:



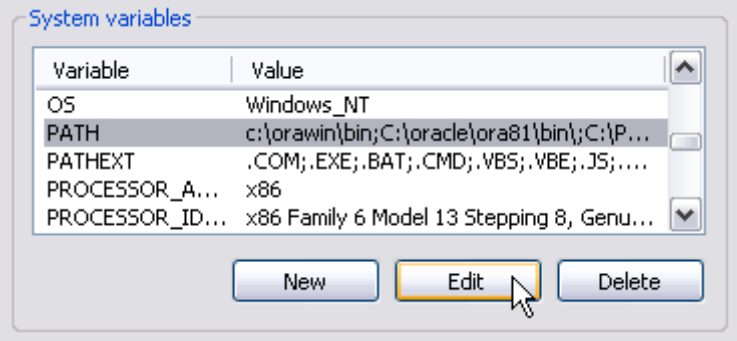
Then enter the variable name and its value:



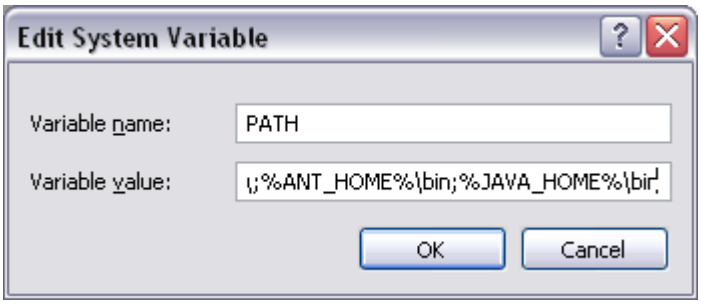
Do the same thing for ANT_HOME variable:



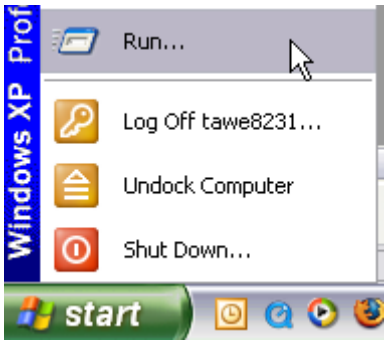
Now modify the PATH variable. Select the PATH variable and click on the Edit button:

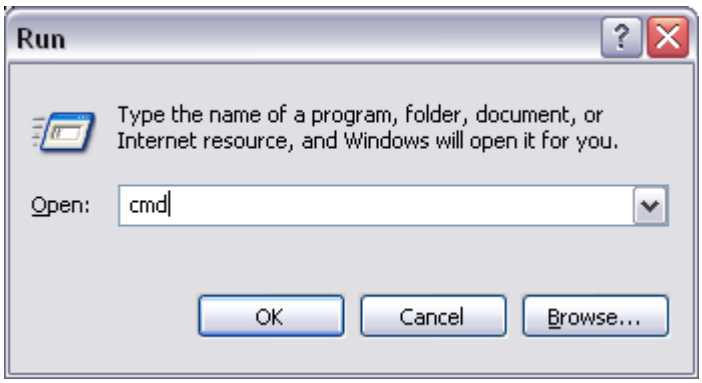


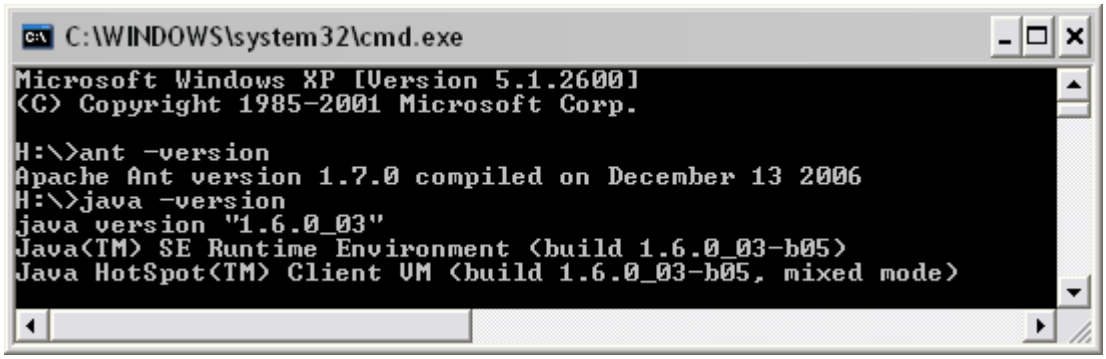
Now add the reference to the ANT_HOME\bin and JAVA_HOME\bin repertory at the end of the PATH line:



Now you just have to check if the good java version and ant version are used by your system.







3.2.2. Linux OS

Now you have to set the following environment variable:

- JAVA_HOME=/usr/lib/jdk1.6.0_03
- ANT_HOME=/usr/lib/apache-ant-1.7.0

And to modify:

- PATH=$PATH:$HOME/bin:$ANT_HOME/bin:$JAVA_HOME/bin

Go to the root directory of your user account.

Modify the .bash_profile file with the following command line: vi .bash_profile

If this file doesn't exist you can create it.

Put in it the following line:

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
# User specific environment and startup programs
ANT_HOME=/usr/lib/apache-ant-1.7.0
JAVA_HOME=/usr/lib/jdk1.6.0_03
PATH=$PATH:$HOME/bin:$ANT_HOME/bin:$JAVA_HOME/bin

export ANT_HOME
export JAVA_HOME
export PATH
unset USERNAME
```

Now you have to logg off from your platform and then logg on to reload the .bash_profile file.

Now you just have to check if the good java version and ant version are used by your system.

```
[clif@        ~]$ ant -version
Apache Ant version 1.7.0 compiled on December 13 2006
[clif@        ~]$ java -version
java version "1.6.0_02"
Java(TM) SE Runtime Environment (build 1.6.0_02-b05)
Java HotSpot(TM) Server VM (build 1.6.0_02-b05, mixed mode)
[clif@        ~]$
```

3.2.3. Mac OS X

[TODO]

Appendix A: system properties

A number of Java system properties are set in file etc/clif.props of CLIF runtime environment. This file is used by the helper ant targets provided in file build.xml located at the root of CLIF runtime environment. Should you need to use CLIF without ant, don't forget to set all these system properties when launching the appropriate class in your Java Virtual Machine.

System properties used by CLIF are listed in the table hereafter:

system property	comment	default value in file etc/clif.props	default value in binary code
java.security.policy	set Java security policy file	etc/java.policy	*none*
fractal.provider	set Fractal implementation	org.objectweb.fractal.julia.Julia	*none*
fractal.registry.host	set hostname running FractalRMI registry. The registry is now integrated to the console (so the host is the console's host)	localhost	
fractal.registry.port	set port number for the FractalRMI registry launched by the console.	1234	
julia.config	using Julia as Fractal implementation, set Julia configuration file	etc/julia.cfg	*none*
julia.loader	using Julia as Fractal implementation, set class loader	org.objectweb.fractal.julia.loader.DynamicLoader	*none*
clif.codeserver.port	set port number for class and resource server embedded in the console	1357	*none*
clif.codeserver.host	set host name for class and resource server embedded in the console	localhost	*none*

clif.codeserver.path	ordered set of directories where the codeserver may look for classes and resources it is asked for, separated by ; character. Note that, whatever the value of this property, classes and resources are first looked for in the jar files in lib/ext/ directory, and in the console's current directory. Absolute paths are used as is, while relative paths are interpreted from the root of CLIF's runtime environment.	examples/classes/ *(just to make examples run)*	*none*
clif.datacollector.delay_s	Sets the delay (in seconds) before writing an event to the storage system. Typical value should be greater than the variation of response times to get events stored in chronological order.	10	10
clif.filestorage.dir	Sets the file system directory to be created (if necessary) and used to store the generated measures. An absolute path is used as is, while a relative path is interpreted from the root of CLIF's runtime environment.	*none*	report
clif.isac.threads	Size of ISAC execution engine's pool of thread. The optimal value depends on the average requests throughput and the average response time.	10	10
clif.isac.groupperiod	update period (in ms) of active virtual users populations to match the specified load profiles	100	100
clif.isac.schedulerperiod	polling period (in ms) for the threads of the thread pool asking for something to do	1	1
clif.isac.jobdelay	When positive, gives the delay threshold (in ms) before an alarm is generated when a think time is longer than specified. -1 disables this feature.	-1	-1

clif.filestorage.host	sets a local IP address or a subnet number to be elected by the filestorage component when collecting events through TCP/IP sockets	*commented out*	*random choice among locally available*
jonathan.connectionfactory.host	sets a local IP address or a subnet number to be used by the FractalRMI remote object references	*commented out*	*random choice among locally available*

Other system properties may be useful for a variety of use cases (they are given in comments in file `etc/clif.props.template`):

- for remote Java debugging:
  `-agentlib:jdwp=transport=dt_socket,address=8000,server=y,suspend=n`

  for SSL certificates (for example for HTTPS support):

```
-Djavax.net.ssl.trustStore=/path/to/keystore
-Djavax.net.ssl.trustStorePassword=the_keystore_password
```

Appendix A: XML DTDs for ISAC

ISAC scenarios (org/ow2/clif/scenario/isac/dtd/scenario.dtd)

```
<!-- A scenario is composed of two parts:-->
<!--  - behaviors, to define some behavior...-->
<!--  - load, to define the load repartition...-->
<!ELEMENT scenario (behaviors,loadprofile)>
<!-- In the part behaviors, we must define the plugins that will be used in behaviors-->
<!ELEMENT behaviors (plugins,behavior+)>
<!-- For each plugin we define the plugin with the use tag-->
<!ELEMENT plugins (use*)>
<!-- We can add some parameters if it's needed-->
<!ELEMENT use (params?)>
<!-- We define an id which can be used in the next parts, to reference the plugin used-->
<!-- The name is the name of the plugin that will be used-->
<!ATTLIST use
  id        ID     #REQUIRED
  name      CDATA  #REQUIRED
>
<!-- Now we can define the behaviors-->
<!-- a behavior begin with the behavior tag, and can be composed of: -->
<!--   - A sample: reference to a specified sample plugin... -->
<!--   - A timer: it's a reference to a timer plugin... -->
<!--   - A while controller: it's a while loop... -->
<!--   - A preemptive: it's a controller adding a preemptive for all it children... -->
<!--   - An if controller: it's a controller doing the if / then /else task... -->
<!--   - A nchoice controller: it's a controller which permits doing random choices between
some sub-behaviors with a weight factor -->
<!ELEMENT behavior (sample|timer|control|while|preemptive|if|nchoice)*>
<!-- When we define a behavior we must define the id parameter too, -->
<!-- it will be used to reference behavior in load part-->
<!ATTLIST behavior
  id        ID     #REQUIRED
>
<!-- A sample element could need some parameters-->
<!-- the parameters needed are defined in the plugin, which will be used, definition file-->
<!ELEMENT sample (params?)>
<!-- A sample element have for parameter: -->
<!--  - use: the id of the plugin that will be used for this sample-->
<!--         the id of this plugin must be defined into the plugins part-->
<!--  - name: the name of the action that is referenced by the sample tag-->
<!--          this action name must be specified in the plugin, which is used, definition file-->
<!ATTLIST sample
  use       CDATA  #REQUIRED
  name      CDATA  #REQUIRED
>
<!-- A timer element could need some parameters-->
<!-- the parameters needed are defined in the plugin, which will be used, definition file-->
<!ELEMENT timer (params?)>
<!-- The timer have got the same parameters of a sample element-->
<!ATTLIST timer
  use       CDATA  #REQUIRED
  name      CDATA  #REQUIRED
>
<!ELEMENT control (params?)>
<!ATTLIST control
```

```
 use      CDATA  #REQUIRED
 name     CDATA  #REQUIRED
>
<!-- A while controller must contain a condition and a sub-behavior-->
<!ELEMENT while (condition,(sample|timer|control|while|preemptive|if|nchoice)*)>
<!-- A condition is a reference to a test of a specified plugin-->
<!-- it could need some parameters-->
<!ELEMENT condition (params?)>
<!-- we need specified as parameters for this tag, the plugin used and the name of the test,
like sample or timer tag-->
<!ATTLIST condition
 use      CDATA  #REQUIRED
 name     CDATA  #REQUIRED
>
<!-- A preemptive element is defined as a while element, the difference is in the execution
process-->
<!-- For a while we evaluate the condition before each loop, in a preemptive before each
action...-->
<!ELEMENT preemptive (condition,(sample|timer|control|while|preemptive|if|nchoice)*)>
<!-- An if controller must contains a condition and a sub-behavior ('then' tag)-->
<!-- And optionally it could contain another sub-behavior ('else' tag)-->
<!ELEMENT if (condition,then,else?)>
<!-- A then tag delimited the sub-behavior that will be executed if the condition is true-->
<!ELEMENT then (sample|timer|control|while|preemptive|if|nchoice)*>
<!-- A else element contains a sub-behavior too-->
<!ELEMENT else (sample|timer|control|while|preemptive|if|nchoice)*>
<!-- A nchoice plugin contains n sub-behavior, each sub-behavior have a probability to be
executed-->
<!ELEMENT nchoice (choice+)>
<!-- An choice element contain a sub-behavior-->
<!ELEMENT choice (sample|timer|control|while|preemptive|if|nchoice)*>
<!-- And this element take for parameter a probability-->
<!ATTLIST choice
 proba    CDATA  #REQUIRED
>
<!-- Now we define the params element, this element begin the part to define parameters for
the parent element-->
<!ELEMENT params (param+)>
<!-- For each param we need to define it with the param tag-->
<!ELEMENT param EMPTY>
<!-- This tag take for parameters the name of the parameter and it value-->
<!ATTLIST param
 name     CDATA  #REQUIRED
 value    CDATA  #REQUIRED
>
<!-- Now let's define the load part, this part is used to define the ramps, each ramps represent
the load for a behavior-->
<!-- We can define some ramps together in a group element, this element is used to launch
several behaviors in the same time-->
<!ELEMENT loadprofile (group*)>
<!-- A group is a composition of 'ramp' elements-->
<!ELEMENT group (ramp+)>
<!-- We need define the behavior id of the group and optionally -->
<!-- the force stop mode, default is true -->
<!ATTLIST group
 behavior       CDATA       #REQUIRED
 forceStop                  (true|false) "true"
```

```
>
<!-- each ramp could take some parameters-->
<!ELEMENT ramp (points)>
<!-- For a ramp we must define the style of the ramp, which will be used-->
<!ATTLIST ramp
 style     CDATA  #REQUIRED
>
<!ELEMENT points (point,point)>
<!ELEMENT point EMPTY>
<!-- For a ramp we must define the style of the ramp and the reference of the behavior, which
will be used-->
<!ATTLIST point
 x     CDATA  #REQUIRED
 y     CDATA  #REQUIRED
>
```

ISAC plug-ins declaration (org/ow2/clif/scenario/isac/dtd/plugin.dtd)

```
<!-- A plugin definition begin with the plugin tag, and may contain: -->
<!-- samples, timers, tests and a session object-->
<!ELEMENT plugin (object,(sample|timer|control|test)*,help?)>
<!-- We must define the name of the plugin -->
<!ATTLIST plugin
 name     CDATA  #REQUIRED
>
<!-- We could define an object session -->
<!-- The session object could need some parameters to be initialized-->
<!ELEMENT object (params?,help?)>
<!-- We must define the id of the object and the class of the object-->
<!ATTLIST object
 class    CDATA  #REQUIRED
>
<!-- We can define some samples -->
<!-- The sample could need some params-->
<!ELEMENT sample (params?,help?)>
<!-- We now define the sample name, the sample class,-->
<!ATTLIST sample
 name     CDATA  #REQUIRED
 number   CDATA  #REQUIRED
>
<!-- We could define some tests-->
<!ELEMENT test (params?,help?)>
<!-- We have the name of the test,-->
<!--  and the number of the test stored in the session object -->
<!ATTLIST test
 name    CDATA  #REQUIRED
 number  CDATA  #REQUIRED
>
<!-- A timer have a same functioning as a test element -->
<!-- takes a name to be referenced in a behavior, and must define the number of the timer to
be used -->
<!ELEMENT timer (params?,help?)>
<!ATTLIST timer
 name    CDATA  #REQUIRED
 number  CDATA  #REQUIRED
>
<!ELEMENT control (params?,help?)>
<!ATTLIST control
```

```
 name     CDATA  #REQUIRED
 number   CDATA  #REQUIRED
>
<!-- Definition of the parameters element, contains several param-->
<!ELEMENT params (param+)>
<!ELEMENT param EMPTY>
<!-- For each param we define the name and the type of the parameter, -->
<!-- that will be asked when the user will used the parent element in a behavior-->
<!ATTLIST param
 name     CDATA  #REQUIRED
 type     CDATA  #REQUIRED
>
<!-- We can define an help for the plugin, or for a sample or an object... -->
<!ELEMENT help (#PCDATA)>
```

ISAC plug-ins GUI aspects (org/ow2/clif/scenario/isac/dtd/gui.dtd)

```
<!ELEMENT gui (object,(sample|test|timer|control)*)>
<!ELEMENT object (params)>
<!ATTLIST object
 name       CDATA        #REQUIRED
>
<!ELEMENT sample (params)>
<!ATTLIST sample
 name       CDATA        #REQUIRED
>
<!ELEMENT test (params)>
<!ATTLIST test
 name       CDATA        #REQUIRED
>
<!ELEMENT timer (params)>
<!ATTLIST timer
 name       CDATA        #REQUIRED
>
<!ELEMENT control (params)>
<!ATTLIST control
 name       CDATA        #REQUIRED
>
<!ELEMENT params (param|group)*>
<!ELEMENT param (radiobutton|field|checkbox|nfield|combo|table)>
<!ATTLIST param
 name     CDATA        #REQUIRED
 label    CDATA        #IMPLIED
>
<!ELEMENT group (param|group)*>
<!ATTLIST group
 name     CDATA        #REQUIRED
>
<!ELEMENT radiobutton (choice*)>
<!ELEMENT choice EMPTY>
<!ATTLIST choice
 value    CDATA        #REQUIRED
 default  (true|false)  "false"
>
<!ELEMENT checkbox (choice*)>
<!ELEMENT field EMPTY>
<!ATTLIST field
 size     CDATA        #REQUIRED
```

```
 text     CDATA        ""
>
<!ELEMENT nfield EMPTY>
<!ELEMENT table EMPTY>
<!ATTLIST table
 cols     CDATA        #REQUIRED
>
<!ELEMENT combo (choice*)>
```