December, 11th 2009

HOW TO

Develop an ISAC PLUGIN for CLIF v2

with the ECLIPSE RCP console

http://clif.ow2.org/

Table of contents

1. Introduction to ISAC Plug-ins....3
1.1. What is an ISAC Plug-ins....3
1.2. Objective of this tutorial....3
2. LDAP directory overview....4
2.1.1. Object Tree Structure....4
2.1.2. Attributes....4
2.1.3. ObjectClass....4
3. How to Write a LDAP Injector ISAC Plug-in....6
3.1. How to Create an Isac Plug-in project....6
3.2. Writing your ISAC plug-ins....11
3.2.1. Descriptor files overview....11
3.2.2. The SessionObject Class overview....13
3.2.3. LDAP injector primitives....14
3.2.4. Session object....16
3.2.5. Adding Samples....19
3.2.6. Complete SessionObject class with Java code....24

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, DHCP, 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. Objective of this tutorial

This tutorial will guide you on how to create an Isac plug-in from scratch. You will see how to create a LDAP injector that can be used in a Isac scenario. Once you completed the plug-in creation, you will be able to use it as specified in the Clif tutorial.

2. LDAP directory overview

LDAP is a lightweight directory access protocol described in RFC 2251-2256,2829-2830, It defines a lightweight access mechanism in which clients send requests to and receive responses from LDAP servers.

LDAP enabled directories use a data model that **assumes** or **represents** the data as a hierarchy of objects. This does not imply that LDAP is an object-oriented database. As pointed out above, LDAP itself is a protocol that allows access to an LDAP enabled service and does not define how the data is stored - but the operational primitives (read, delete, modify) operate on a model (description) of the data that has object-like characteristics (mostly).

2.1.1. Object Tree Structure

Data are represented in an LDAP enabled directory as a hierarchy of objects, each of which is called an entry. The resulting tree structure is called a Data Information Tree (DIT). The top of the tree is commonly called the root (a.k.a **base** or **suffix**).

Each **entry** in the tree has one parent entry (object) and one or more child entries (objects). Each child entry (object) is a sibling of its parent's other child entries.

Each entry is composed of (is an instance of) one or more objectClass. **ObjectClasses** contain zero or more attributes. Attributes have names (and sometimes abbreviations or aliases) and typically contain data (at last!).

Summary:

1. Each **Entry** is composed of one or more **objectClasses**
2. Each **objectClass** has a name and contains **Attributes**.
3. Each **Attribute** has a name, usually contains data and is a member of an **object class**.

2.1.2. Attributes

Each **attribute** has a name and normally contains data. **Attributes** are always associated with (are members of) one or more **ObjectClasses**. **Attributes** have a number of interesting characteristics:

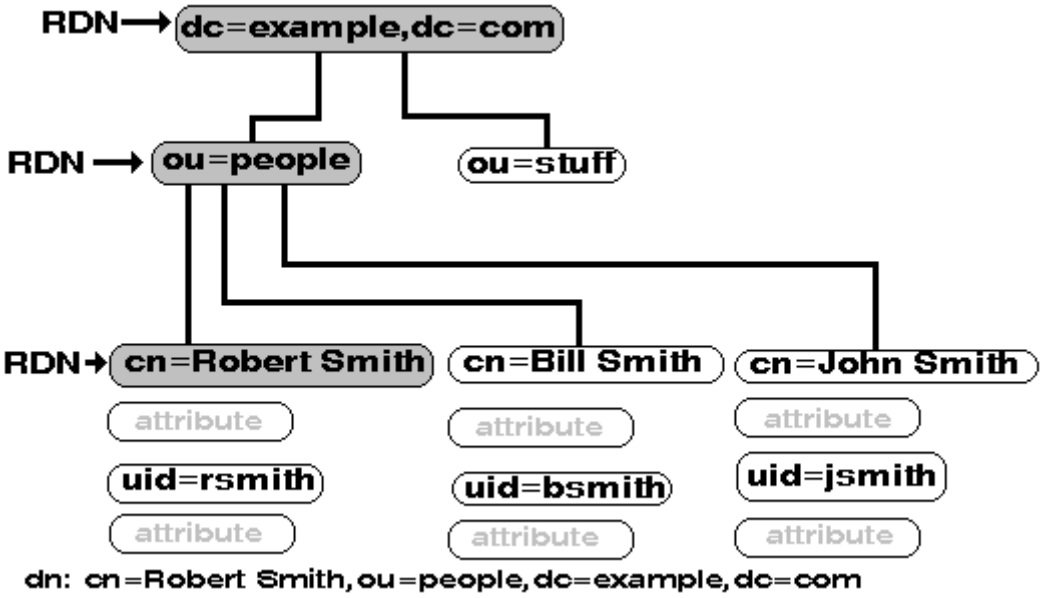
1. All **attributes** are members of one or more **objectclass(es)**
2. Each **attribute** defines the data type that it may contain.
3. **Attributes** can be optional or mandatory,
4. **Attributes** can have single or multi values,
5. **Attributes** have names and sometimes aliases or abbreviations,
6. At each level in the hierarchy the data contained in one **attribute** should uniquely identify the **entry**. It can be any **attribute** in the entry. It can even be a combination of two or more attributes.

2.1.3. ObjectClass

ObjectClasses are essentially packages of **attributes**. There are a confusing number of pre-defined **objectClasses**, each of which contains bucket-loads of **attributes** for almost all common or garden applications. But of course the one you NEED is never defined! **objectclasses** have two more characteristics:

1. The **objectclass** defines whether an attribute member **MUST** (mandatory) be present or **MAY** (optional) be present.

2. The **objectclass** may be part of a hierarchy in which case it **inherits** all the characteristics of its parent **objectclasses**.



Example of an LDAP Tree

3. How to Write a LDAP Injector ISAC Plug-in

Writing your own ISAC plug-in is a simple way to customize the injection capabilities of ISAC, still relying on the generic language for defining behaviors and load profiles. Writing an ISAC plug-in basically consists in defining a Java class that encapsulates (a part of) the state of each behavior instance, and provides specific methods for:

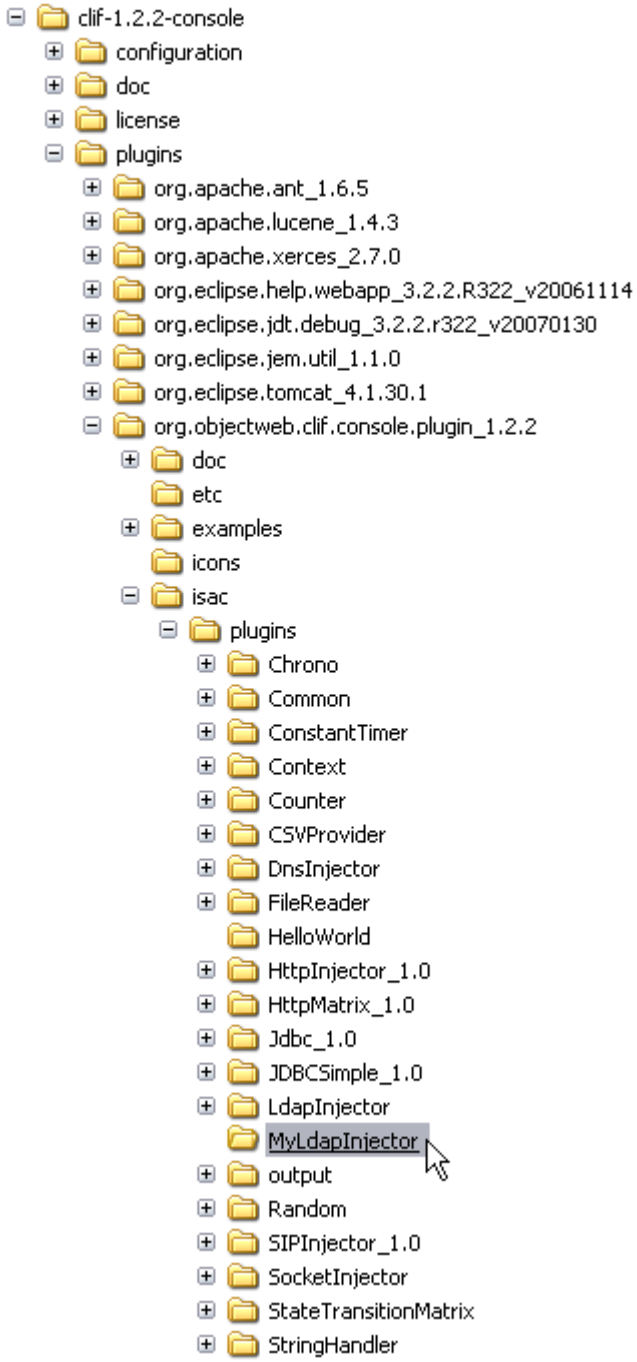
- instantiating new *session objects* for new behavior instances;
- implementing load injection primitives;
- implementing timer primitives (e.g. to implement think times);
- implementing external data provisioning;
- implementing condition primitives;
- session object control primitives.

The primitives offered by an ISAC plug-in, as well as a GUI-oriented description for its parameters, are declared through 3 descriptor files:

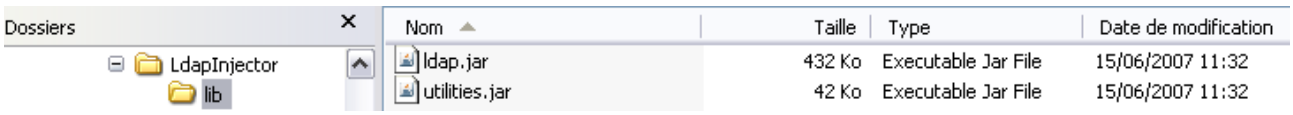
- plugin.properties specifies Java properties plugin.name, plugin.xmlFile and plugin.guiFile to respectively set the ISAC plug-in name, the name of the XML file describing the list of primitives and parameters, and the name of the XML file describing the GUI concerns. Usual values for these file names respectively are plugin.xml and gui.xml.
- plugin.xml (or any other name as specified in plugin.properties file)
- gui.xml (or any other name as specified in plugin.properties file)

3.1. How to Create an Isac Plug-in project

To add a new ISAC plug-in, you must create a directory in subdirectory isac/plugins of the CLIF execution environment. For our example we will create a MyLdapInjector directory.

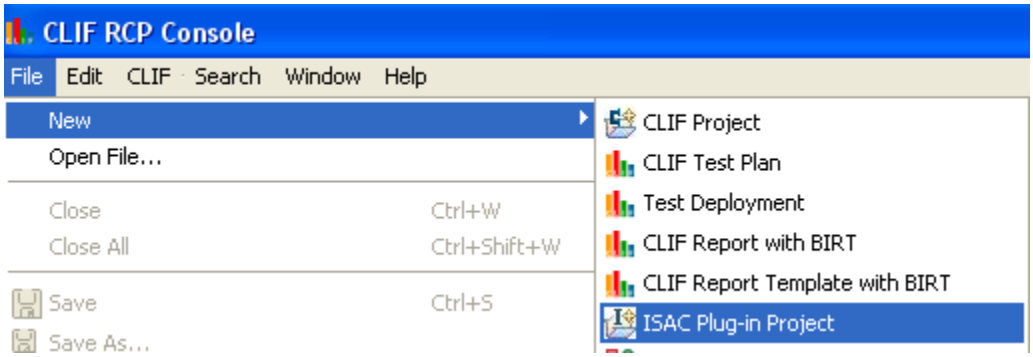


To be able to act on an LDAP directory we need some specific libraries. You have to get ldap.jar and utilities.jar. You can get these librairies into the LdapInjector/lib directory, Once you have them, create repertory lib in MyLdapInjector and paste them into it.



From the Eclipse-RCP standalone console you can use the ISAC plug-in creation Wizard:

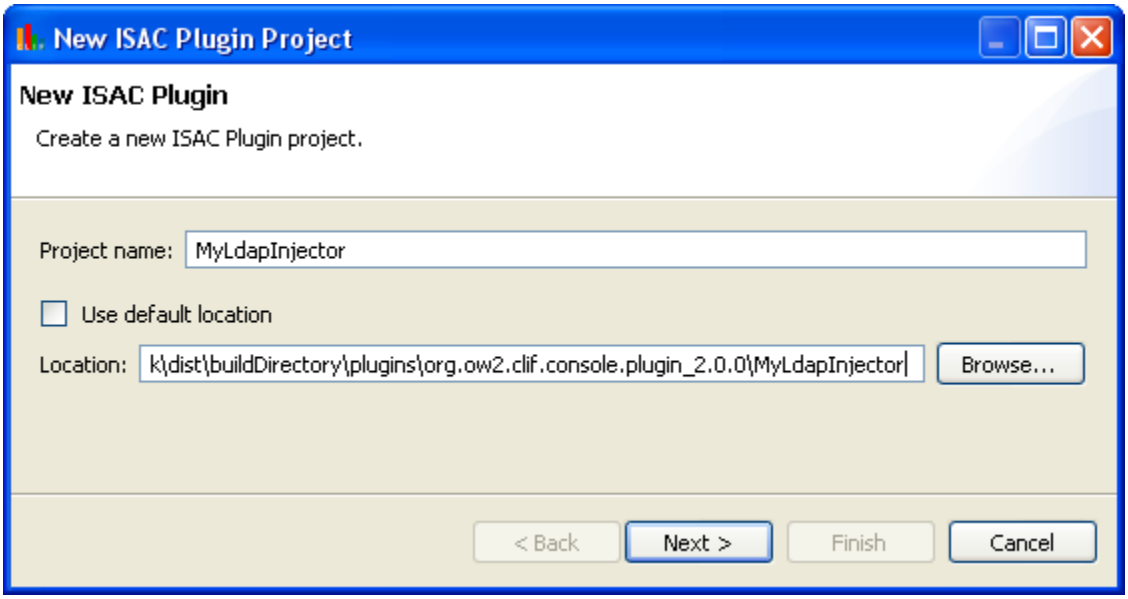
Click on: File > New > ISAC Plug-in Project



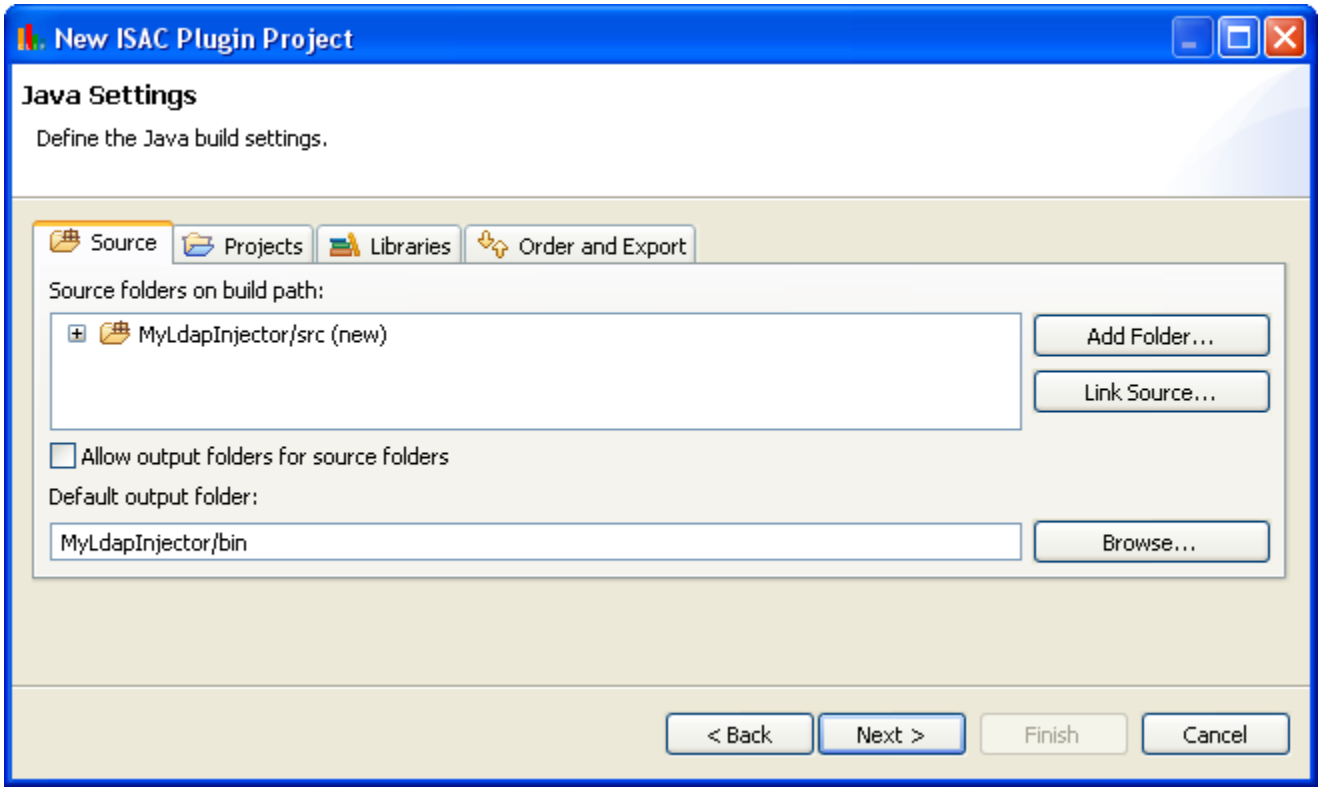
Enter your project name: in our case MyLdapInjector

Enter your project location (refer to the repertory that you created before). In our case the location is:

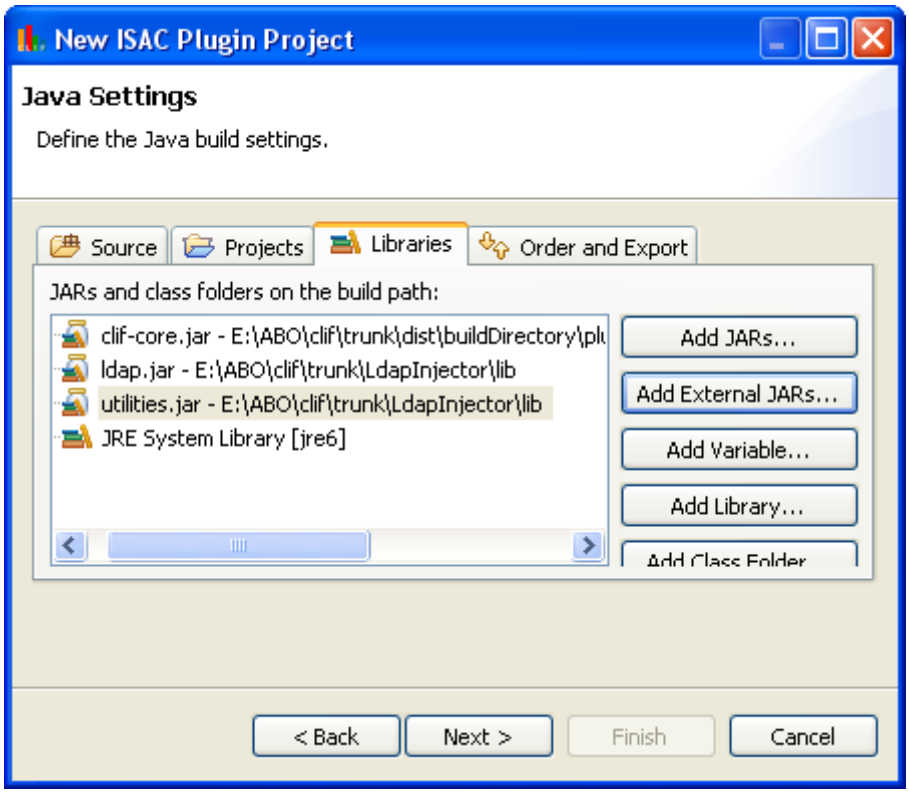
/path/to/the/eclipseStandAloneConsole/clif-<versions>-eclipseconsole/plugins/org.ow2.clif.console.plugin_2.0.0/isac/plugins/MyLdapInjector



Click on the Next button.

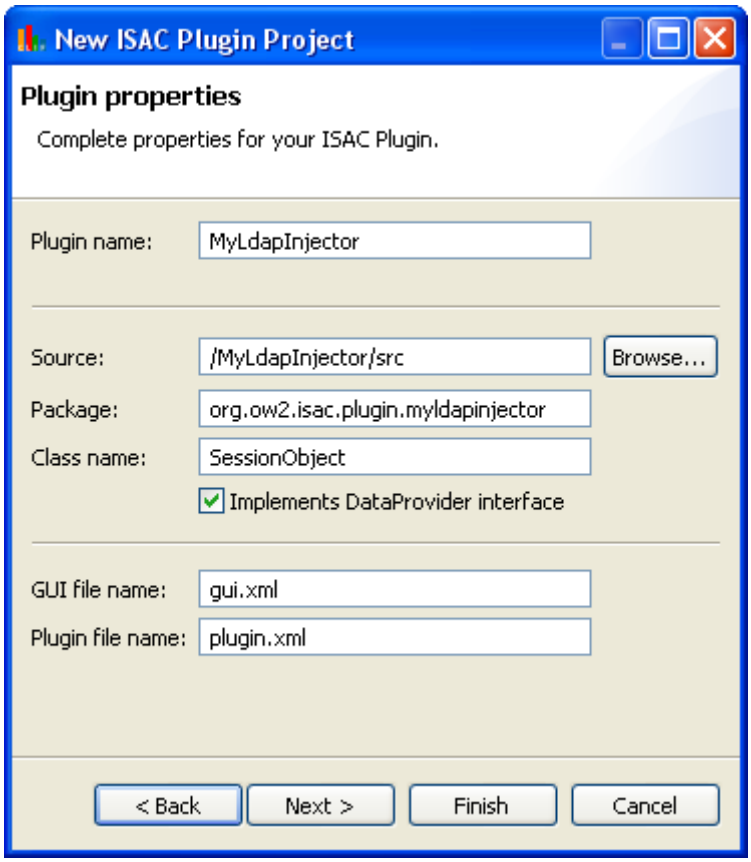


In the Java settings, you have to add LDAP libraries (ldap.jar and utilities.jar) to your classpath.



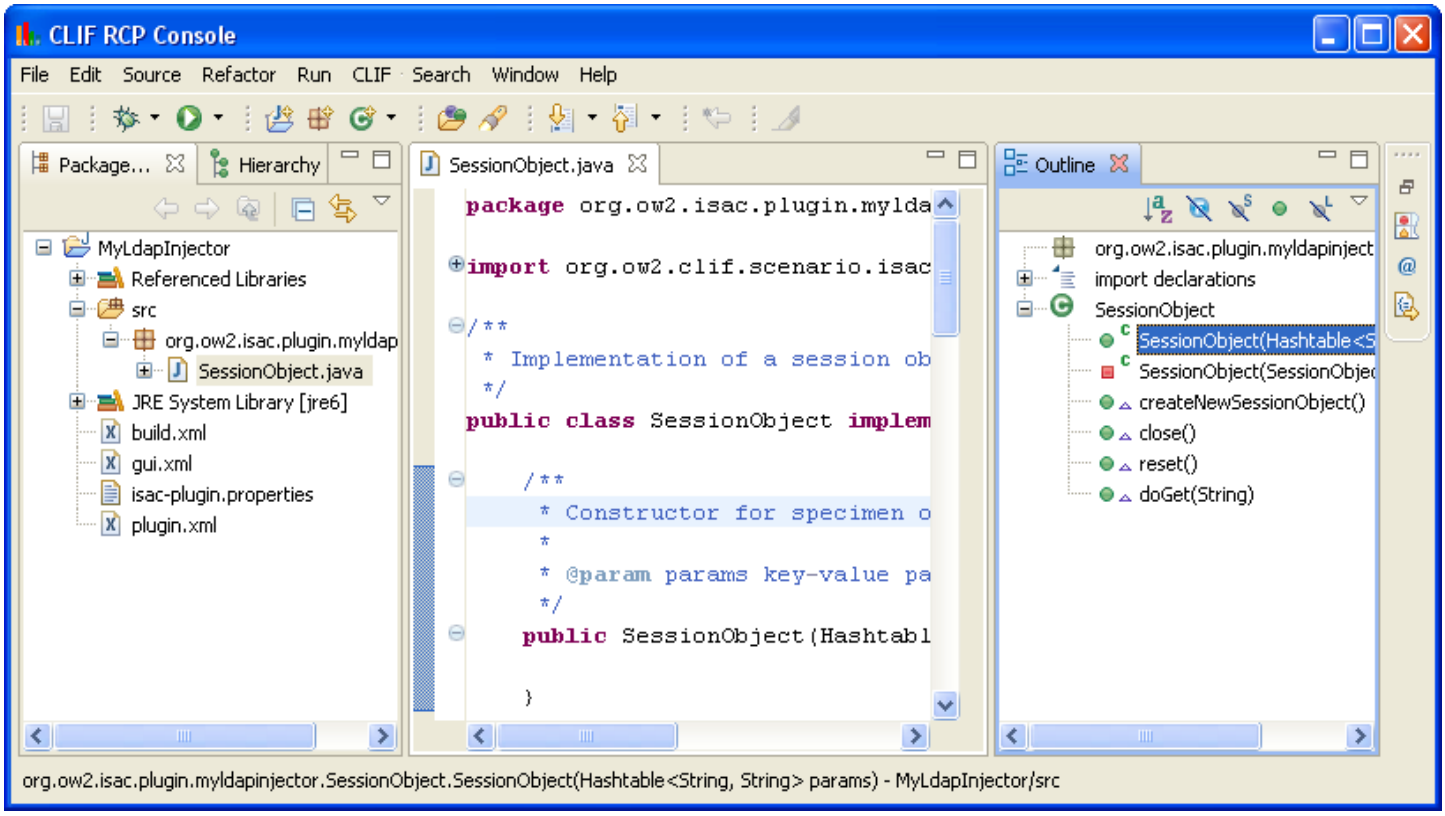
Click on the Next button.

Set the plug-in properties. Most of them have default values.



You will need to tick the Implements DataProvider interface box if your plug-in have to provide data.

You can now click on the Finish button. The Eclipse Isac perspective will be loaded. It is recommended to click on Finish now to be sure that all settings will be saved in case of an Eclipse crash.



3.2. Writing your ISAC plug-ins

3.2.1. Descriptor files overview

The plugin.properties files:

```
plugin.name=MyLdapInjector
plugin.guiFile=gui.xml
plugin.xmlFile=plugin.xml
```

The plug-in descriptor file specifies (plugin.xml by default):

- the plug-in name, which must match the plug-in's directory name,
- the associated session object class and the initial settings parameters, with some help
- the samples, controls, conditions and timers with their parameters and help.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugin PUBLIC "-//objectweb.org//DTD CLIF Isac 1.0//EN"
"classpath:org/objectweb/clif/scenario/isac/dtd/plugin.dtd">

<plugin name="MyLdapInjector">
  <object class="org.objectweb.isac.plugin.myldapinjector.SessionObject">
    <params></params>
    <help>Give object help.</help>
  </object>
</plugin>
```

The user interface descriptor file (gui.xml by default) adds explicit labels to primitives and parameters, and associates each parameter to GUI-related information. Possible graphical

widgets are available through the following tags : radiobutton, field, checkbox, nfield (variable number of fields), combo. Parameters may also be visually grouped together with the group tag. The parameter value resulting from a nfield widget is the concatenation of the variable number of fields separated by one ';' character.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gui PUBLIC "-//objectweb.org//DTD CLIF IsacGUI 1.0//EN"
"classpath:org/objectweb/clif/scenario/isac/dtd/gui.dtd">

<gui>
  <object name="SessionObject">
    <params></params>
  </object>
</gui>
```

The Eclipse wizard also creates a build.xml file which will be used when you will use the ant isac-clean and ant isac-plugins tasks of the build.xml file present in the following directory:

clif-<version>-consoleplugins\org.ow2.clif.console.plugin_<plugin_version>

This XML file contains several ant tasks:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ISAC-plugin_MyLdapInjector" default="compile">
<!-- This build.xml file must be called from CLIF main build.xml file. 4
properties are provided by CLIF main build.xml file:
      - libext.dir, all Jars used by the plugin must be copied to this
directory
      - clif.classpath, classpath for full CLIF runtime library
      - build.dir, build directory (where classes must be generated and
necessary resource files,
      if any, must be copied)
-  isac.dir, ISAC root directory -->
-
  <!-- General Configuration -->
  <property
      name="plugin.dir" value="${isac.dir}/plugins/MyLdapInjector">
  </property>
  <!-- classpath definition -->
  <path id="plugin.compile.classpath">
    <pathelement path="${clif.classpath}"></pathelement>
    <pathelement path="${build.dir}"></pathelement>
  </path>
  <target name="compile">
    <javac
      srcdir="${plugin.dir}/src" destdir="${build.dir}"
      classpathref="plugin.compile.classpath">
    </javac>
    <copy todir="${build.dir}/MyLdapInjector">
      <fileset
            dir="${plugin.dir}"
            includes="plugin.properties,plugin.xml,gui.xml">
      </fileset>
    </copy>
  </target>
  <target name="clean"></target>
</project>
```

3.2.2. The SessionObject Class overview

The last file generated by the Eclipse wizard is the SessionObject.java file. This Java class implements the SessionObjectAction interface to handle replication of specimens for creation of session objects that will be actually associated to behavior instances (method createNewSessionObject()). This interface is also used for freeing resources used by session objects before they are discarded (method close()), and recycling old session objects into fresh ones (method reset()).

```
package org.objectweb.isac.plugin.myldapinjector;

import org.objectweb.clif.scenario.isac.plugin.SessionObjectAction;
import java.util.Hashtable;
import org.objectweb.clif.scenario.isac.plugin.DataProvider;
import org.objectweb.clif.scenario.isac.exception.IsacRuntimeException;

/**
 * Implementation of a session object for plugin ~MyLdapInjector~
 */
public class SessionObject implements SessionObjectAction, DataProvider {

  /**
   * Constructor for specimen object.
   *
   * @param params key-value pairs for plugin parameters
   */
  public SessionObject(Hashtable params) {}

  /**
   * Copy constructor (clone specimen object to get session object).
   *
   * @param so specimen object to clone
   */
  private SessionObject(SessionObject so) {}

  ////////////////////////////////////////
  // SessionObjectAction implementation //
  ////////////////////////////////////////

  /**
   * @see
org.objectweb.clif.scenario.isac.plugin.SessionObjectAction#createNewSessionObj
ect()
   */
  public Object createNewSessionObject() {
      return new SessionObject(this);
  }

  /**
   * @see org.objectweb.clif.scenario.isac.plugin.SessionObjectAction#close()
   */
  public void close() {}

  /**
   * @see org.objectweb.clif.scenario.isac.plugin.SessionObjectAction#reset()
   */
  public void reset() {}
```

```
  ////////////////////////////////
  // DataProvider implementation //
  ////////////////////////////////

  /**
   * @see org.objectweb.clif.scenario.isac.plugin.DataProvider#doGet()
   */
  public String doGet(String var) {
      throw new IsacRuntimeException("Unknown parameter value in
~MyLdapInjector~ ISAC plugin: " + var);
  }
}
```

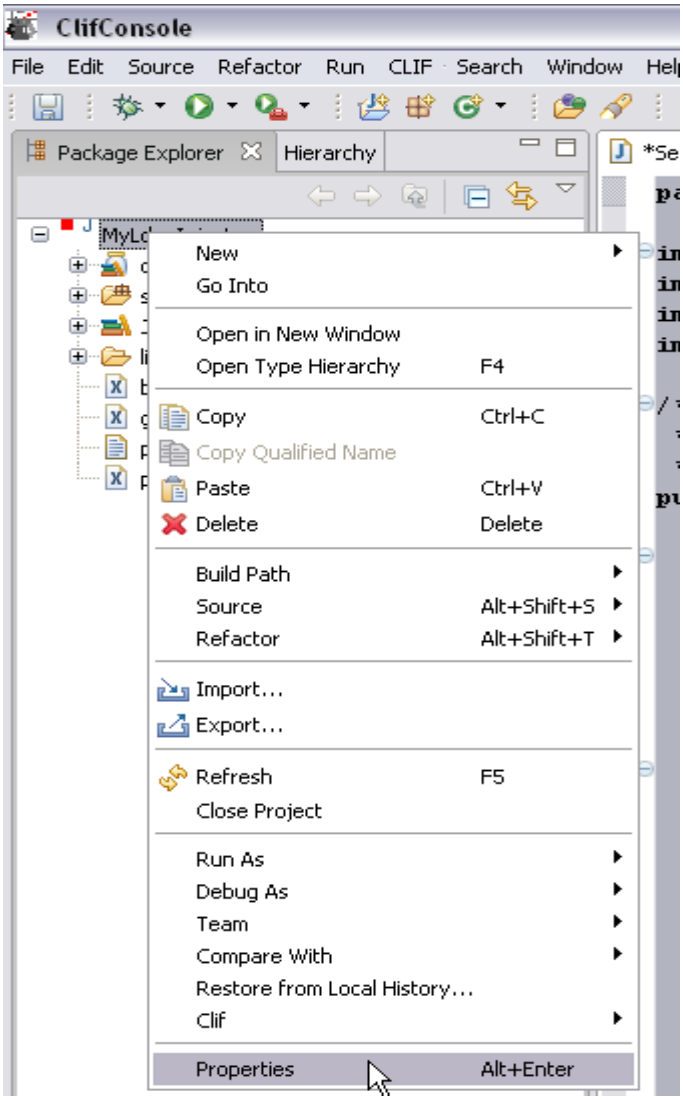
3.2.3. LDAP injector primitives

Now we can define all the primitives of our LDAP injector. In this tutorial only two primitives will be define:

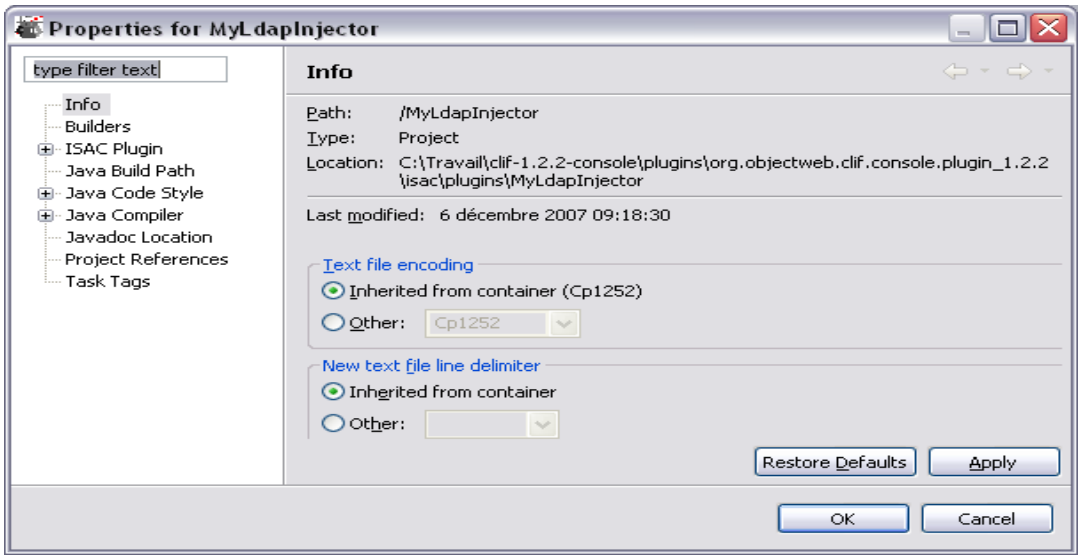
- Connection: connection to the LDAP directory (bind)
- CloseConnection: close the connection to the LDAP directory (unbind)

To define these primitives we will use the Eclipse wizard.

Right click on the MyLdapInjector project, then click on properties:

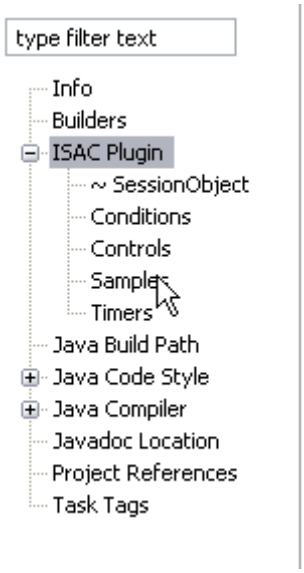


The following window appears:



Now with the ISAC Plugin menu you will be able to define:

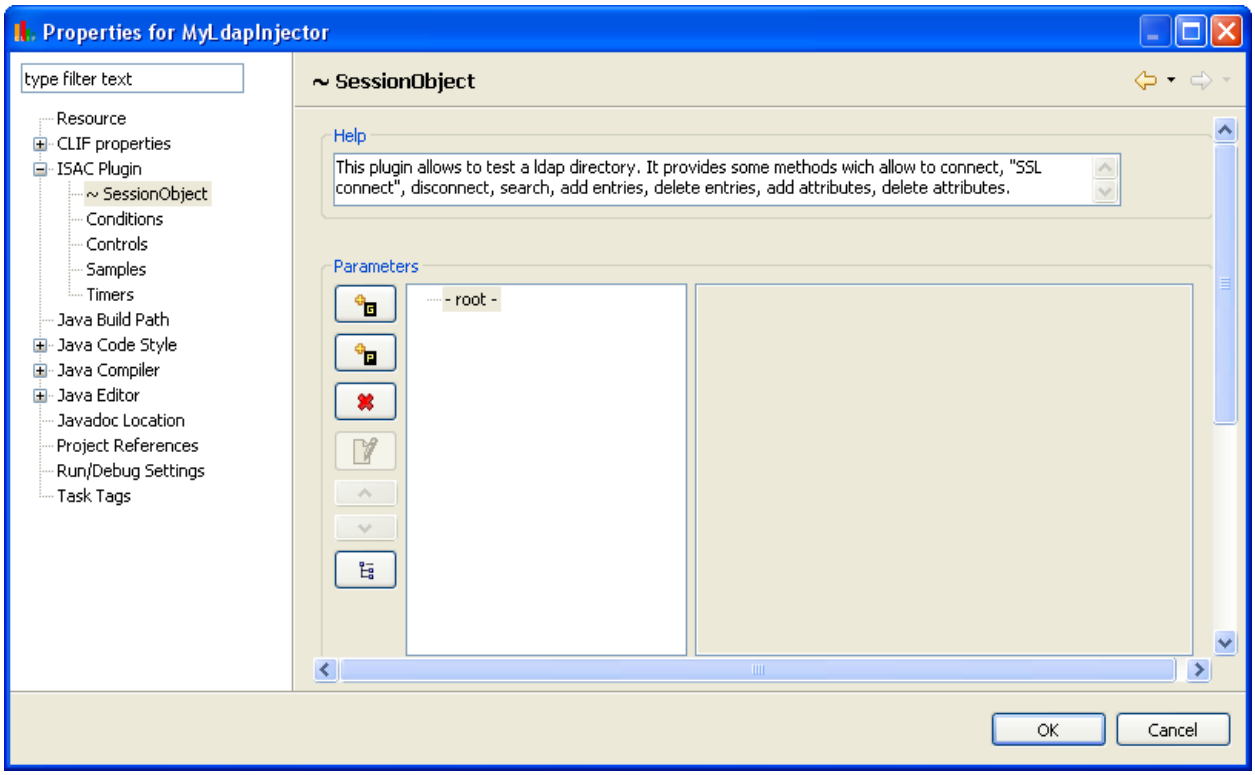
- SessionObject
- Conditions
- Controls
- Sample
- Timers



3.2.4. Session object

In the session object, global attributes of the LdapInjector will be defined.

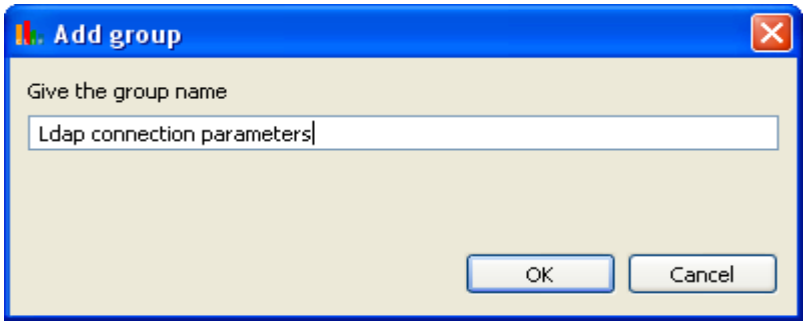
First of all, you have to write the definition of your LDAP injector. This definition will be displayed in the help contextual menu when you will define your ISAC scenario.



After adding contextual help you can define groups of global parameters. For example, you can define a group that contain severals parameters.

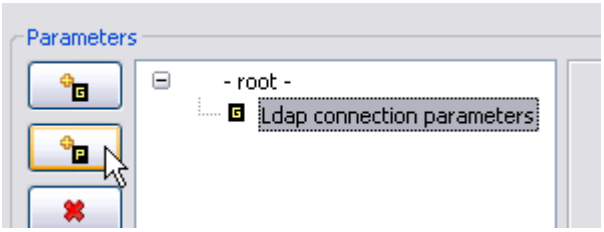
Click on  to add a group in the session object.

Give the group name:

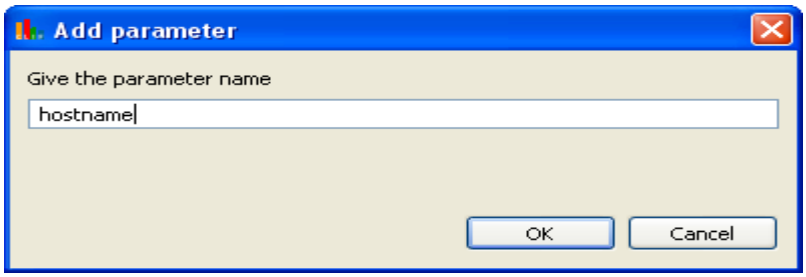


Then you can add several parameters into the group that you have created.

First of all, select it, and then click on 



Give the parameter name:

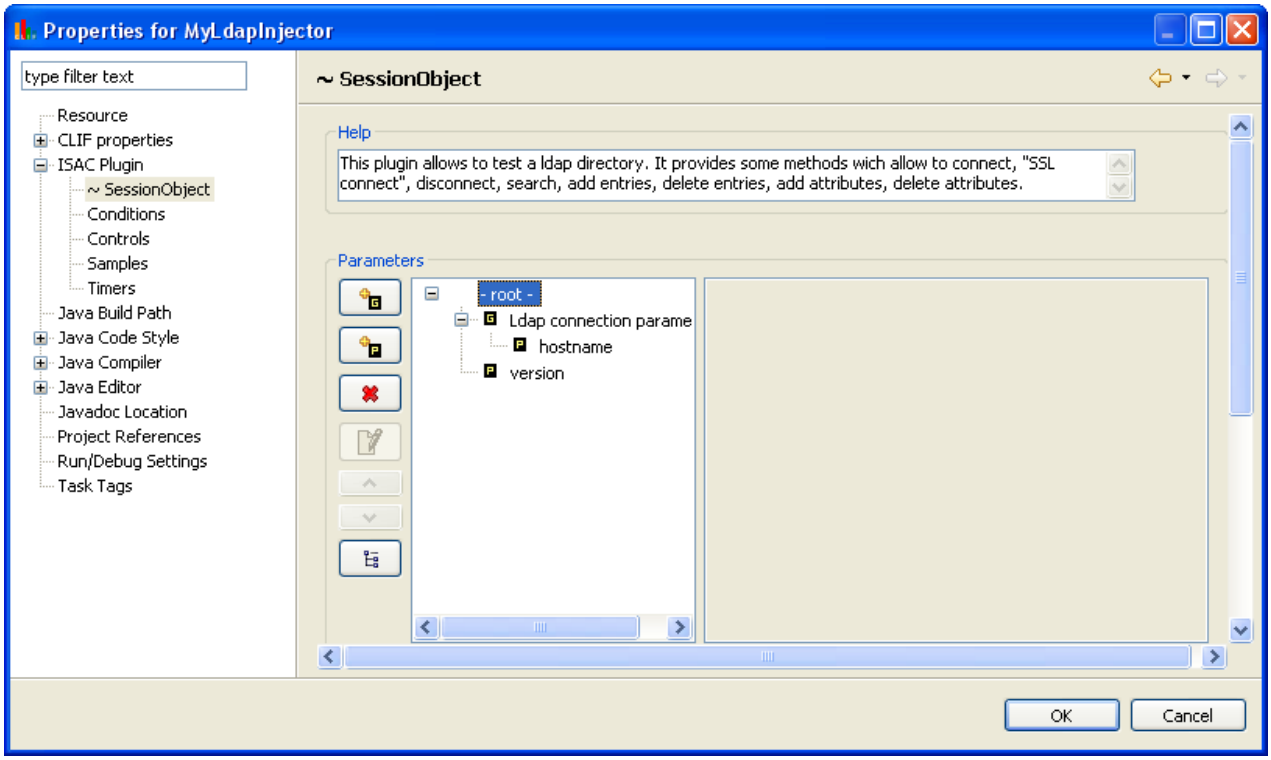


Do it with all your parameters.

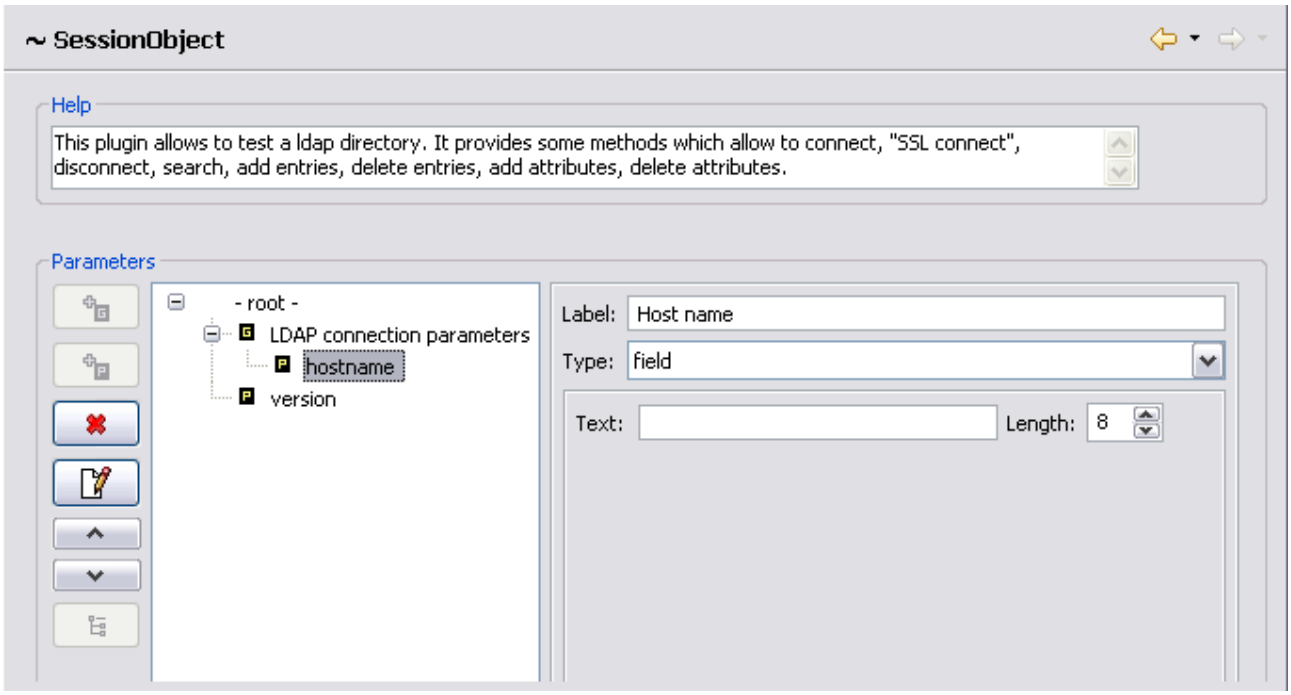
If you want to add parameters which are not connection parameters you have to click first on “root” and then on 

Finally you will have something like that:

Now you have to give each parameter a name, a type, a default text value if necessary, a list of values, depending of the parameters you are defining.



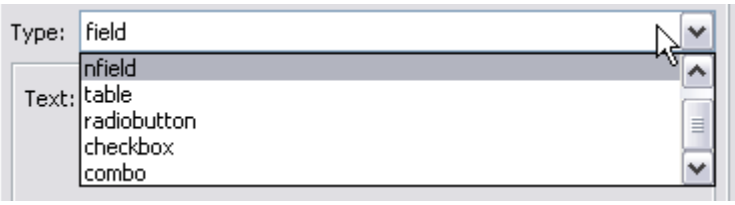
Click on the parameter you want to define:



To choose the parameter's type there is a list of choice:

- field

- nfield
- table
- radio button
- checkbox
- combo



In your case (LDAP injector) you will choose field type for all parameters.

So you just have to define a Label (this will be the label displayed when you will write your ISAC scenario), the type, a text if our parameter has a default value and a length (this length represents the size of the input text field of the Isac scenario wizard).

These actions with the wizard modify:

- SessionObject.java class adding attributes:

```
    static final String PLUGIN_VERSION = "version";
    static final String PLUGIN_HOSTNAME = "hostname";
```

- plugin.xml file adding parameters:

```
  <params>
    <param name="hostname" type="String"></param>
    <param name="version" type="String"></param>
  </params>
```

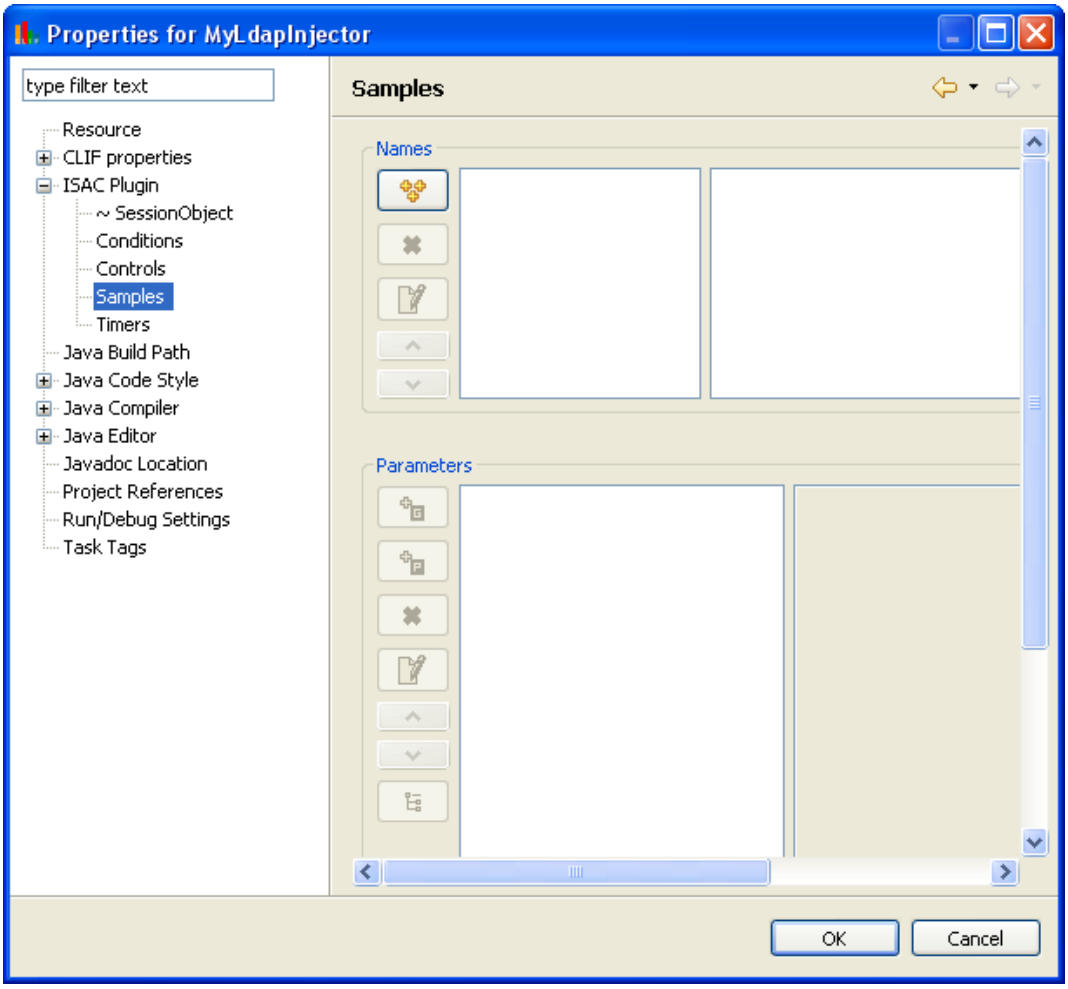
- gui.xml files adding parameters:

```
  <params>
    <group name="Ldap connection parameters">
      <param label="Host name" name="hostname">
        <field text="" size="8"></field>
      </param>
    </group>
    <param label="LDAP version" name="version">
      <field text="" size="8"></field>
    </param>
  </params>
```

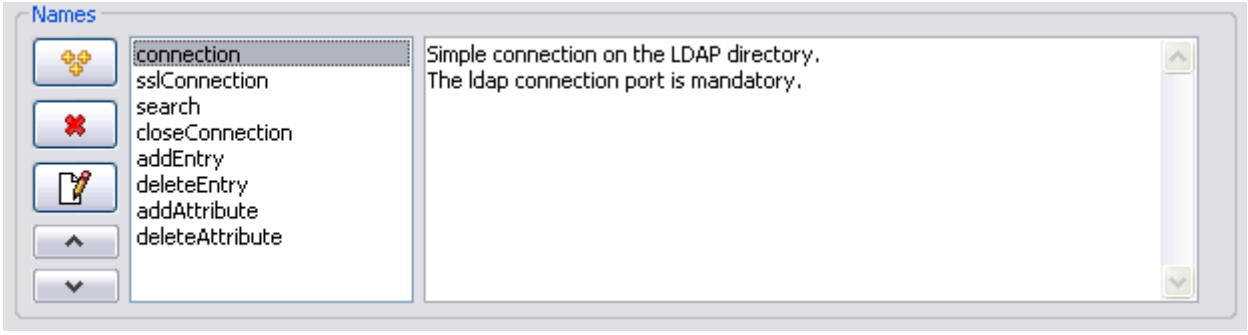
3.2.5. Adding Samples

To add samples is to define primitives of LDAP injector. These primitives are:

- connection
- SSL connection
- closeConnection
- search
- add entry
- delete entry
- add attribute
- delete attributes



Click on the  button and add all the primitives you want to define giving them a name and don't forget to write a description of all of them.



After adding the primitive's name and description you will have to add their specific attributes. Each primitive has specific attributes:

- connection
  - login: the login DN to connect to the LDAP directory
  - password: the password to connect to the LDAP directory
  - port: the port to connect using non-secure connection.
- sslConnection

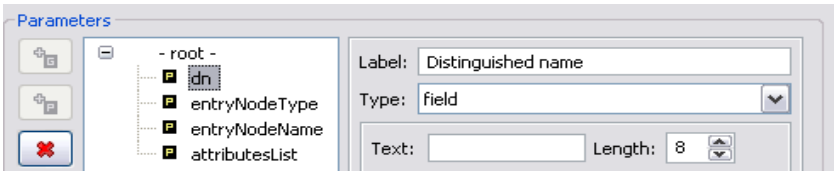
  - login: the login DN to connect to the LDAP directory
  - password: the password to connect to the LDAP directory
  - port: The port to connect using secure connection.
  - keystorePath: The path to access the keystore file.
- closeConnection (no attributes)
- search
  - searchBase: the place in the directory tree where to start to searching. (e.g. ou=people,dc=orange,dc=fr)
  - searchFilter: the search filter (e.g. sn=fran*)
  - searchScope: the search depth. (e.g. 2)
- addEntry
  - dn: the LDAP distinguished name
  - entryNodeType: type of the node to insert (cn or sn or ou ou or uid)
  - entryNodeName: name of the node to insert
  - attributesList: node's list of attributes
- deleteEntry
  - dn: distinguished name representing the node to delete.
- addAttribute
  - dn: distinguished name of the node where we want to add attributes
  - attributesList: list of the couple attributes name / attributes values
- deleteAttribute
  - dn: distinguished name of the node where we want to delete attributes
  - attributestodelete: name of the attributes to delete.

Now we focus on the definition of the attributes of the addEntry primitive.

To an add attribute just click on  the add parameter button (notice that you can define group of attributes as with session object).

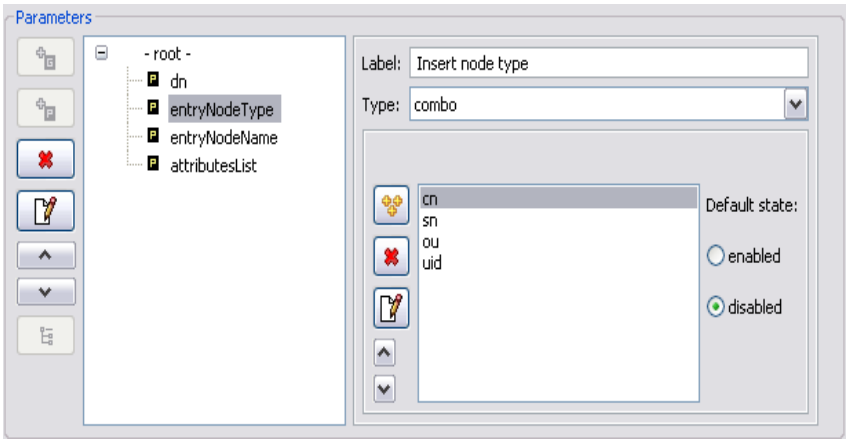
The dn attribute has:

- label: Distinguished name
- type: field
- text: no default value
- length: 8



The entryNodeType has:

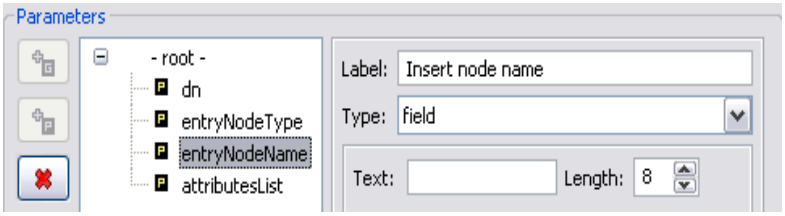
- label: Insert node type
- type: combo



To add values to combobox click on  and enter the different values with a default state (enabled or disabled).

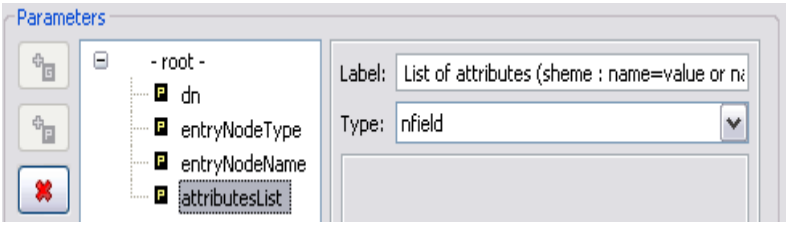
The entryNodeName has:

- label: Insert node name
- type: field
- text: no default text
- length: 8



The attributesList has:

- label: List of attributes (scheme: name=value or name=value1,value2,value3)
- type: nfield



These actions with the wizard modify these files:

- SessionObject.java class
- gui.xml
- plugin.xml

This variable SAMPLE_ADDENTRY identifies the addEntry primitive in the LdapInjector plug-in. Each primitive has unique identifier.

```
    static final int SAMPLE_ADDENTRY = 4;
```

The variables below are added when you define addEntry attributes. Each attributes of each primitive is identified with a variable which value is the variable name defined in the wizard.

```
    static final String SAMPLE_ADDENTRY_ATTRIBUTESLIST = "attributesList";
    static final String SAMPLE_ADDENTRY_ENTRYNODENAME = "entryNodeName";
    static final String SAMPLE_ADDENTRY_ENTRYNODETYPE = "entryNodeType";
    static final String SAMPLE_ADDENTRY_DN = "dn";
```

Adding samples with the wizard modify this Java class by adding a doSample() method which returns an ActionEvent object. This method has three parameters:

- The first argument gives the primitive identifier.
- The second parameter gives the list of parameter values indexed by their names, as set in the plug-in descriptor file using tag params;

- The third argument gives a report object whose fields will have to be filled before being returned.

Basically, the doSample() method is supposed to perform a load injection request, wait for some kind of response, state if this request is a success or a failure, measure its response time and return a sample report. Returning null is also possible, to make CLIF ignore this sample.

```
   /**
   @see org.objectweb.clif.scenario.isac.plugin.SampleAction#doSample()
   */
   public ActionEvent doSample(int number, Map params, ActionEvent report) {
   switch (number) {
      case SAMPLE_DELETEATTRIBUTE:
        break;
      case SAMPLE_ADDATTRIBUTE:
        break;
      case SAMPLE_DELETEENTRY:
        break;
      case SAMPLE_ADDENTRY:
        break;
      case SAMPLE_SEARCH:
        break;
      case SAMPLE_SSLCONNECTION:
        break;
      case SAMPLE_CONNECTION:
        break;
      default:
        throw new Error("Unable to find this sample in ~LdapInjector~ ISAC
plugin: " + number);
    }
    throw new IsacRuntimeException("No action defined for this sample in
~LdapInjector~ ISAC plugin: " + number);
     }
```

- plugin.xml:

In the plugin.xml file the addEntry sample is added with all its parameters and its help.

```
  <sample name="addEntry" number="4">
    <params>
      <param name="dn" type="String"></param>
      <param name="entryNodeType" type="String"></param>
      <param name="entryNodeName" type="String"></param>
      <param name="attributesList" type="String"></param>
    </params>
    <help>Add entry into an ldap directory.&#xD;
To be able to add an entry into ldap directory the following parameters are
madatory:&#xD;
    - Distinguished name&#xD;
    - Node type (type of the node to insert) &#xD;
    - Node name (name of the node to insert)&#xD;
    - List of the attributes: couple name=value(s) of the attributes which
composed the node.</help>
  </sample>
```

- gui.xml:

The user interface descriptor file adds explicit labels to primitives and parameters, and associates each parameter to GUI related information. Graphical widgets appears here (combo, field, nfield).

```
  <sample name="addEntry">
    <params>
      <param label="Distinguished name" name="dn">
        <field text="" size="8"></field>
      </param>
      <param label="Insert node type" name="entryNodeType">
        <combo>
          <choice value="cn" default="false"></choice>
          <choice value="sn" default="false"></choice>
          <choice value="ou" default="false"></choice>
          <choice value="uid" default="false"></choice>
        </combo>
      </param>
      <param label="Insert node name" name="entryNodeName">
        <field text="" size="8"></field>
      </param>
      <param label="List of attributes (sheme: name=value or
name=value1,value2,value3)" name="attributesList">
        <nfield></nfield>
      </param>
    </params>
  </sample>
```

3.2.6. Complete SessionObject class with Java code

At this point we have finished using the ISAC plug-in wizard. We are now going to write Java code to implement the LDAP connection (secure or non-secure), disconnection and add entry methods.

When you write an ISAC scenario you can import several times the LdapInjector plug-in with different settings. For each import the ISAC execution engine instantiates and initializes with specific settings a specimen session object. For that purpose, your plug-in class must implement a public constructor taking a Map as a single argument. This Map will hold the specimen settings with the parameters names as keys, as specified in the plug-in XML descriptor file. The specimen objects will be used just for replication, according to the load profiles, but will never be associated to behavior instances.

Here are global parameters which have to be set for every imported LdapInjector plug-in.

```
  private String hostname;
  private int ldapVersion;
```

In the SessionObject constructor we checked the value of the parameters. These must not be null or empty, they must be set.

```
  /**
   * Constructor for specimen object.
   *
   * @param params key-value pairs for plugin parameters
   * @throws ClifException
   */
  public SessionObject(Hashtable params) throws ClifException {
  // local address setting
    String value = (String)params.get(PLUGIN_HOSTNAME);
    if (value != null && value.length() > 0) {
```

```
      try {
        hostname = value;
      } catch (Exception ex) {
        throw new ClifException("ISAC can't get hostname because the specified
hostname is not valid: " + value, ex);
      }
    } else  {
      throw new ClifException("ISAC can't get hostname because the specified
hostname is not valid: " + value);
    }

  // ldap version setting
    value = (String)params.get(PLUGIN_VERSION);
    if (value != null && value.length() > 0) {
      try {
        ldapVersion = Integer.parseInt(value);
      } catch (Exception ex) {
        throw new ClifException("ISAC can't get LDAP version because the
specified LDAP version is not valid: " + value, ex);
      }
    } else {
      throw new ClifException("ISAC can't get LDAP version because the
specified LDAP version is not valid: " + value);
    }
  }

  /**
   * Copy constructor (clone specimen object to get session object).
   *
   * @param so specimen object to clone
   */
    private SessionObject(SessionObject so) {
      this.hostname = so.hostname;
      this.ldapVersion = so.ldapVersion;
    }
```

Declare your LDAPConnection object to be able to access it from every implemented method which will manipulate your LDAP directory. This object will be instantiated in the doConnection() method.

```
      private LDAPConnection ldapConnection;
```

All the methods implement for this LDAP injector have three parameters, like the doSample() method:

- The first argument gives the primitive identifier;
- The second parameter gives the list of parameter values indexed by their names, as set in the plug-in descriptor file using tag params;
- The third argument gives a report object whose fields will have to be filled before being returned.

And return an ActionEvent.

These following methods are called by the doSample() method depending of the primitive identifier and need to be implemented:

- doConnection(int number, Map params, ActionEvent report);
- doSslConnection(int number, Map params, ActionEvent report);
- doAddEntry(int number, Map params, ActionEvent report);

- doDeleteEntry(int number, Map params, ActionEvent report);
- doAddAttribute(int number, Map params, ActionEvent report);
- doDeleteAttribute(int number, Map params, ActionEvent report);
- doSearch(int number, Map params, ActionEvent report);
- doCloseConnection(int number, Map params, ActionEvent report);

```
  /**
   * @see org.objectweb.clif.scenario.isac.plugin.SampleAction#doSample()
   */
  public ActionEvent doSample(int number, Map params, ActionEvent report) {
    switch (number) {
      case SAMPLE_DELETEATTRIBUTE:
        return doDeleteAttribute(number, params, report);
      case SAMPLE_ADDATTRIBUTE:
        return doAddAttribute(number, params, report);
      case SAMPLE_DELETEENTRY:
        return doDeleteEntry(number, params, report);
      case SAMPLE_ADDENTRY:
        return doAddEntry(number, params, report);
      case SAMPLE_CLOSECONNECTION:
        return doCloseConnection(number, params, report);
      case SAMPLE_SEARCH:
        return doSearch(number, params, report);
      case SAMPLE_SSLCONNECTION:
        return doSslConnection(number, params, report);
      case SAMPLE_CONNECTION:
        return doConnection(number, params, report);
      default:
        throw new Error("Unable to find this sample in ~LdapInjector~ ISAC
plugin: " + number);
    }
  }
```

Now we will focus more specifically on the addEntry method.

```
  /**
   * Add an entry into a LDAP directory. The connection to the LDAP directory
   * must be established before doing the doAddEntry.
   *
   * @param number: The number which handles the connection to the ldap
   *  directory.
   * @param report: The ActionReport to update.
   * @param params: A Map containing all the useful variable.
   * @return the report.
   */
   private ActionEvent doAddEntry(int number, Map params, ActionEvent report){
```

First of all we have to declare and set attributes of the addEntry primitive. This parameters are in the params Map passed in parameter of the doAddEntry method. The get method applied on a Map returns an Object so we have to cast it into String to be able to use it.

```
  String dn = (String) params.get(SAMPLE_ADDENTRY_DN);
  String insertNodeName = (String) params.get(SAMPLE_ADDENTRY_ENTRYNODETYPE);
  String insertNodeValue = (String) params.get(SAMPLE_ADDENTRY_ENTRYNODENAME);
  String attributesList = (String) params.get(SAMPLE_ADDENTRY_ATTRIBUTESLIST);
```

Then we create a LDAPAttributeSet which is a collection of LDAPAttribute objects. LDAPAttributeSet may be used to build an entry to be added to a directory.

```
  LDAPAttributeSet attributeSet = new LDAPAttributeSet();
```

The attributesList is a string represented by a sequence of couple name=value or name=value1,value2,value3... Each couple is separated by “;”.

We check if the entry node belongs to the attributesList. If it doesn't, we add it, at the end of the attributesList string:

```
  if (attributesList.indexOf(insertNodeName) == -1) {
      attributesList = attributesList+insertNodeName+"="+insertNodeValue+";";
  }
```

We check if the entry node belongs to the dn. If it doesn't, we add it, at the beginning of the dn string;

```
  if (dn.indexOf(insertNodeName) == -1) {
      dn = insertNodeName+"="+insertNodeValue+","+dn;
  }
```

Then we have to transform the String attributesList into a String[] using the split method.

```
  String[] tabParam_NameValue = attributesList.split(";");
```

Now we can go into the String[] tabParam_NameValue and add LDAPAttribute to the LDAPAttributeSet:

```
String name = null;
String[] values = null;

for (int i = 0; i<tabParam_NameValue.length; i++) {
  name=tabParam_NameValue[i].substring(0, tabParam_NameValue[i].indexOf("="));
  values=(tabParam_NameValue[i].substring(tabParam_NameValue[i].indexOf("=")+1,
                                  tabParam_NameValue[i].length())).split(",");
  attributeSet.add( new LDAPAttribute( name, values ));
}
```

Then we can create an LDAPEntry object with the dn and attributeSet as parameters.

```
  LDAPEntry newEntry = new LDAPEntry( dn, attributeSet );
```

To be able to identify the add entry lines in the report file you can set the type parameter in the report object:

```
  report.type = "ADDENTRY_TYPE";
```

And you also need to set the Date parameter of the report object to be able to set the add entry duration.

```
  report.setDate(System.currentTimeMillis());
```

Now you just have to add the entry in your LDAP directory. To do it, you have to apply the add method to the ldapConnection object which is a global attribute of the SessionObject class and which is initialized in the connection() method.

Wee also have to catch exceptions and to set the result of the add, a comment, state of the add method (successful or not):

```
  try {
    ldapConnection.add( newEntry );

    report.duration = (int) (System.currentTimeMillis());
    report.successful = true;
    report.comment = "Added object: " + dn + " successfully.";

  } catch (LDAPException ex) {
    report.successful = false;
    if (ex.getResultCode() == LDAPException.ATTRIBUTE_OR_VALUE_EXISTS) {
      report.result = ex.toString();
      report.comment = "Failed to add existing attribute.";
    } else {
      report.result = ex.toString();
      report.comment = "Failed to add attribute.";
    }
  } catch (Exception e) {
    report.result = e.toString();
    report.comment = "Failed to add existing attribute.";
  }
```

And finally you have to return the report.

```
  return report;
}
```

To be able to test make a search in an LDAP directory you will need to implement doConnection(), doCloseConnection and search() methods.

Here is the Java code of these methods:

```
/**
 * Does a connection to a LDAP directory (bind).
 *
 * @param number: The number which handles the connection to the ldap
directory.
 * @param report: The ActionReport to update.
 * @param params: A Map containing all the useful variable.
 * @return the report.
 */
public ActionEvent doConnection(int number, Map params, ActionEvent report) {
    ldapConnection = new LDAPConnection();
    String login = (String) params.get(SAMPLE_CONNECTION_LOGIN);
    String password = (String) params.get(SAMPLE_CONNECTION_PASSWORD);

    report.type = "CONNECT_TYPE";
    report.setDate(System.currentTimeMillis());

    try {
        // connect to the server
```

```
        ldapConnection.connect( hostname, Integer.parseInt((String)
params.get(SAMPLE_CONNECTION_PORT)));
        // bind to the server
        ldapConnection.bind(ldapVersion, login, password.getBytes("UTF8"));

        report.duration = (int)(System.currentTimeMillis() - report.getDate());
        report.successful = true;
        report.comment = "Successful bind with server.";

    } catch( LDAPException e ) {
        report.successful = false;
        report.result = e.toString();
        report.comment = "ISAC LdapInjector error occured";
    } catch( UnsupportedEncodingException e ) {
        report.successful = false;
        report.result = e.toString();
        report.comment = "ISAC LdapInjector can't UTF8 encode password
"+password;
    }
    return report;
}

/**
 * Close an opened connection to a LDAP directory (bind).
 *
 * @param number: The number which handles the connection to the ldap
directory.
 * @param report: The ActionReport to update.
 * @param params: A Map containing all the useful variable.
 * @return the report.
 */
public ActionEvent doCloseConnection(int number, Map params, ActionEvent
report)
{
    if (ldapConnection != null) {
        report.type = "DISCONNECT_TYPE";
        report.setDate(System.currentTimeMillis());

        try {
            // disconnect with the server
            ldapConnection.disconnect();
            report.duration = (int)(System.currentTimeMillis() -
report.getDate());
            ldapConnection = null;
            report.successful = true;
            report.comment = "ISAC LdapConnection disconnection Successful";

            return report;
        } catch (LDAPException e) {
            report.successful = false;
            report.comment = "ISAC LdapInjector can't disconnect from " +
hostname;
            report.result = e.toString();
            return report;
        }
    } else {
        report.successful = false;
        report.comment = "ISAC LdapInjector can't disconnect unopen ldap
connection";
```

```
        report.result = "ignored";
        return null;
      }
}

/**
 * Does search into a LDAP directory.
 *
 * @param number: The number which handles the connection to the ldap
directory.
 * @param report: The ActionReport to update.
 * @param params: A Map containing all the useful variable.
 * @return the report.
 */
public ActionEvent doSearch(int number, Map params, ActionEvent report) {
      try {
            report.type = "SEARCH_TYPE";
            report.setDate(System.currentTimeMillis());

            LDAPSearchResults searchResults = ldapConnection.search((String)
params.get(SAMPLE_SEARCH_SEARCHBASE),Integer.parseInt((String)params.get(SAMPLE
_SEARCH_SEARCHSCOPE)),(String)
params.get(SAMPLE_SEARCH_SEARCHFILTER),null,false);

            report.duration=(int)(System.currentTimeMillis()-report.getDate());
            report.successful = true;
            report.comment = "LDAP Search succeded";

      } catch( LDAPException e ) {
            report.successful = false;
            report.result = e.toString();
            report.comment = "ISAC LdapInjector error occured";
      }
      return report; }
```

To be able to compile your ISAC plug-in you have to complete the build.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ISAC-plugin_MyLdapInjector" default="compile">
  <!-- This build.xml file must be called from CLIF main build.xml file.
4 properties are provided by CLIF main build.xml file:
      - libext.dir, all Jars used by the plugin must be copied to this directory
      - clif.classpath, classpath for full CLIF runtime library
      - build.dir, build directory (where classes must be generated and necessary resource files,
      if any, must be copied)
      - isac.dir, ISAC root directory
-->
  <!-- General Configuration -->
  <property name="plugin.dir" value="${isac.dir}/plugins/MyLdapInjector"></property>
  <!-- classpath definition -->
  <path id="plugin.compile.classpath">
    <pathelement path="${clif.classpath}"></pathelement>
    <pathelement path="${build.dir}"></pathelement>
    <fileset dir="${plugin.dir}/lib" includes="*.jar"></fileset>
  </path>
  <target name="compile">
```

```
    <javac srcdir="${plugin.dir}/src" destdir="${build.dir}" classpathref="plugin.compile.classpath"></
javac>
    <copy todir="${libext.dir}" overwrite="yes" preservelastmodified="yes">
      <fileset dir="${plugin.dir}/lib" includes="*.jar"></fileset>
    </copy>
    <copy todir="${build.dir}/MyLdapInjector">
      <fileset dir="${plugin.dir}" includes="plugin.properties,plugin.xml,gui.xml"></fileset>
    </copy>
  </target>
  <target name="clean"></target>
</project>
```

Now you just have to compile and make a jar file with all your ISAC plug-ins.

Go to the following directory:

/path_to_your_clif_console/plugins/org.ow2.clif.console.plugin_<version>

and launch the ant command: ant isac-plugins

Once it is successful you can use your ISAC plug-in.