



Licas Getting Started Guide Version 1.5

[A guide on how to install and quickly run an application on the licas framework]

Kieran Greer, Email:

kgreer@distributedcomputingsystems.co.uk.

<http://distributedcomputingsystems.co.uk/licas.aspx>

Table of Contents

1	Introduction to the Licas System	3
2	Installing the Software	4
3	Quick Start with the Instant Messenger	5
3.1	Install the Licas GUI Application.....	5
3.2	Start the GUI and Server	5
3.3	Add a ServiceFactory Object	8
3.4	Open a Service GUI Interface	9
3.5	Using the Message Service GUI.....	11
4	File Service	12
5	Running a Server	13
6	Running a Client	14
7	Restful-Style Invocation of the Server	14
8	Running the Problem Solver	15
9	Additional Platform Requirements	16
10	User Guides	16

1 Introduction to the Licas System

The licas system is an open source framework for building service-based networks, similar to what you would do on a Cloud or SOA platform. The framework comes with a server for running the services on, mechanisms for adding services to the server, mechanisms for linking services with each other, and mechanisms for allowing the services to communicate with each other. The default communication protocol inside of licas itself is an XML-RPC mechanism, but dynamic invocation of external Web Services is also possible. The download packages also come with an all-in-one GUI that can be used either to test systems or as a practical platform on which to run your own real system. The system is also peer-to-peer, with the client GUI also acting as a server that can be invoked. Licas can therefore be used in one of two different ways. Because of resource constraints, these have been packaged together in the one GUI application.

When you open the GUI application, two of the Module buttons will be visible. These are for starting or running a server and for adding services to it. You can also use just these module panels, and open an application interface to any service; therefore using the system as a platform-independent Cloud or SOA operating system. The menu also allows you to select other more scientific modules that allow for testing certain aspects of service-based networks. This is where the linking, metadata and autonomous features would be most useful. The other user guides and related papers give further details on that. This quick start guide describes how to open and run the message service, which is provided as an example of how to use the system as a ready-made application. The system however is still primarily a platform for programmers, to develop their own services with. There are not a large number of ready-made services, but this example shows how easy it is to write one and the lightweight design and in-built intelligent features might make this package attractive for certain applications.

The licas packages can now be downloaded in a number of different zip files. You do not need to compile anything to use these, just unzip the contents into an empty folder. The current version comes with two different server platforms, one client application and one problem solving package. There are two releases of the server code, which is essentially what the licas system is. One is for the standard J2SE platform, but this has now been re-written to be compatible with the Java J2ME platform as well. The source code is also provided however and can be compiled into a standard Java J2SE application or the mobile CDC application. The CDC platform also required a number of additional add-on jars that can be found on public sites. You may find however that if you try to create an application for an actual mobile phone, you will need to add other packages, as it is only the NetBeans mobile phone emulator that the J2ME version has been tested on.

There is also a setup installer for windows that will install licas and its related files, for using the GUI as an application or platform-independent OS, without any additional programming. The installer can be downloaded from the distributedcomputingsystems.co.uk/licas.aspx web site instead and is an attempt at a more complete product for general use.

The rest of the document is organised as follows: Section 2 describes how to install the software. Sections 3 and 4 describe how to use an installed version to run a service application. The other sections then describe how to use the batch files to start each package separately. Section 5 describes how to run a server, while section 6 describes how to run the client GUI. Section 7 describes how to access the server through a web service interface instead of Java GUI, while section 8 describes how to run a problem solver. Section 9 describes any additional jars, etc, that might be required for developing the software further, while section 10 summarises the contents of the other user guides.

2 Installing the Software

To install the software, just unzip the files into an empty directory. You will find folders for each of the two server compilations, a client and a problem solver. To use any jar executable, double click on the `.bat` or `.sh` file in its folder to start it running. Before loading any services, you firstly need to have a server running somewhere, which can also be locally in the client GUI itself. Alternatively, this can be a separate server on the same machine, or on a different machine. To start a separate server, you only need to run the `runServer.bat` batch file in its directory. To start the GUI, you need to run the `runGui.bat` batch file in the client directory. If you are using Linux/Unix, then you can run the script files (`.sh`) instead, but the GUI is possibly not suitable for Linux, as it requires a Windows standard of screen resolution.

The installer installs into the path `'/DCS/Licas'` in your root user directory. The GUI currently reads other jar files from its associated `'lib'` folder and so installing into the usual `'Program Files'` folder requires administrator privileges. For the time being, it will be installed into your user directory. This means that after using the installer, you will have a folder called `'DCS/Licas'` and a folder called `'licasData'`, both in your root user directory. The first one contains the jar executable and the second one is for data files and the default service classes. Uninstalling will remove both of these folders, but if new content is added, it should be left and not deleted.

3 Quick Start with the Instant Messenger

A default instant messenger service is provided as part of the installation package. It allows you to see how easy it is to run this sort of application and what the required operations are. It might be preferable to simply install the windows setup installer, if you just want to use the GUI application and not program with it. The installer also installs a default data folder in your root user directory. This has the name 'licasData' and it is where the application will look for default configuration information, or the default services package.

A service is represented graphically by a node in the Server panel of the GUI. This is just a view, where the service itself could be running on the GUI server, or on some other remote server. If you right-click on the node, you can then send some commands to the service through this view. Communication with the service itself can be carried out through messages that are converted into text-based streams. It is therefore also possible to add a GUI interface to the service view and again communicate through text-based messages; to ask the service to perform any of its operations and retrieve the results for displaying. In this way, the service can be turned into a complete application, running in a local or remote virtual environment and with an interactive interface for easy use. The following sections describe how to do this.

3.1 Install the Licas GUI Application

To use the system simply for running applications, you can install using the windows installer. Simply start the installer and answer yes to all questions. The installer will add the system to a folder on your Startup menu under the DCS name, and as a shortcut icon on your desktop. Note that the installer will also add a folder called 'licasData' to your root user directory. When the GUI starts, it will automatically look for this folder for its config information. Any default services are also stored there.

3.2 Start the GUI and Server

The GUI can be started either by double clicking on the desktop icon, or through the Startup menu option DCS/Licas/Licas. A Server panel can be accessed by clicking on the Server module button. It should also be the first panel that is displayed. If you click on the Start Server button, this should start a server running locally in the GUI itself. There is a Server tab in the bottom right quadrant that gives output of what the server is doing. There is also a Refresh View toolbar button that will immediately update the network

view when it is clicked. The `Server Details` section of the panel should display the `ip` address that the server is running on. Figure 1 gives an example of the GUI server running.

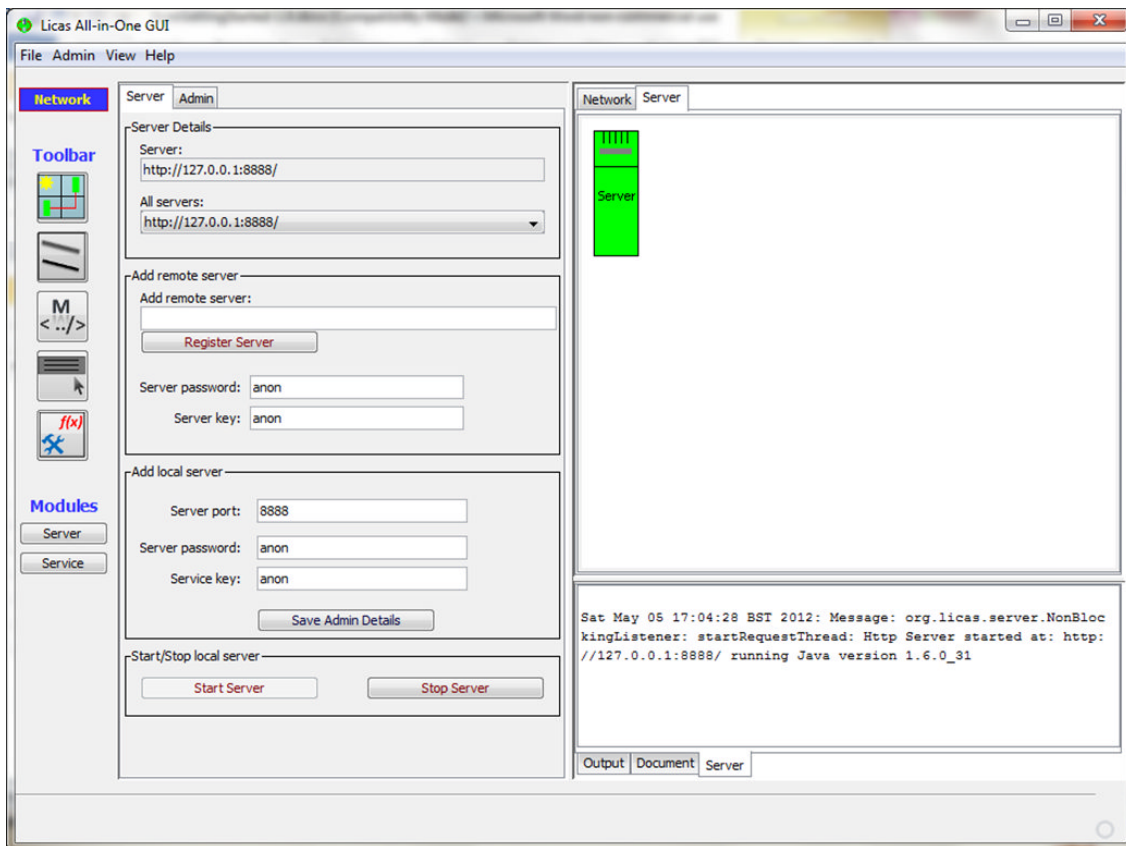


Figure 1. Server panel and tab with server started.

You will then want to load a service onto the network. This can also be done manually with a small amount of effort. If you click the `Service` module button, a `Load Service` panel should be displayed. This provides options to allow you to add your own services to the network, but also allows you to add one of the default services. The bottom half of this panel has an `Add New Service` group of components. These also contain a list of default services. There is a combo box with a number of service type names, for example, `File` or `Message`. Select the `Message` service type from this combo box. Any service needs to be added to another service. If it is a base service, then it needs to be added to the server. To indicate the server, click on the server component in the graphic view. It looks like a green computer with the word 'Server' written on it. This should add the word 'HttpServer' to the `Selected network service` text field. As each service is a Java class, it can be initialised by different constructor types and the default licas classes also allow for this. However, you can simply add the default services as they are. You need to

assign to your service a unique name or id. For a message service, this could be your own name. In the `New service ID` text box, type in the name to assign to the service. If you then click the `Add to Network` button, you should get a message confirming that the service has been added to the server. If you then click the `Refresh network view` toolbar button (the top button) the service should be displayed in the server view. Figure 2 is an example of a message service node being displayed in the server view.

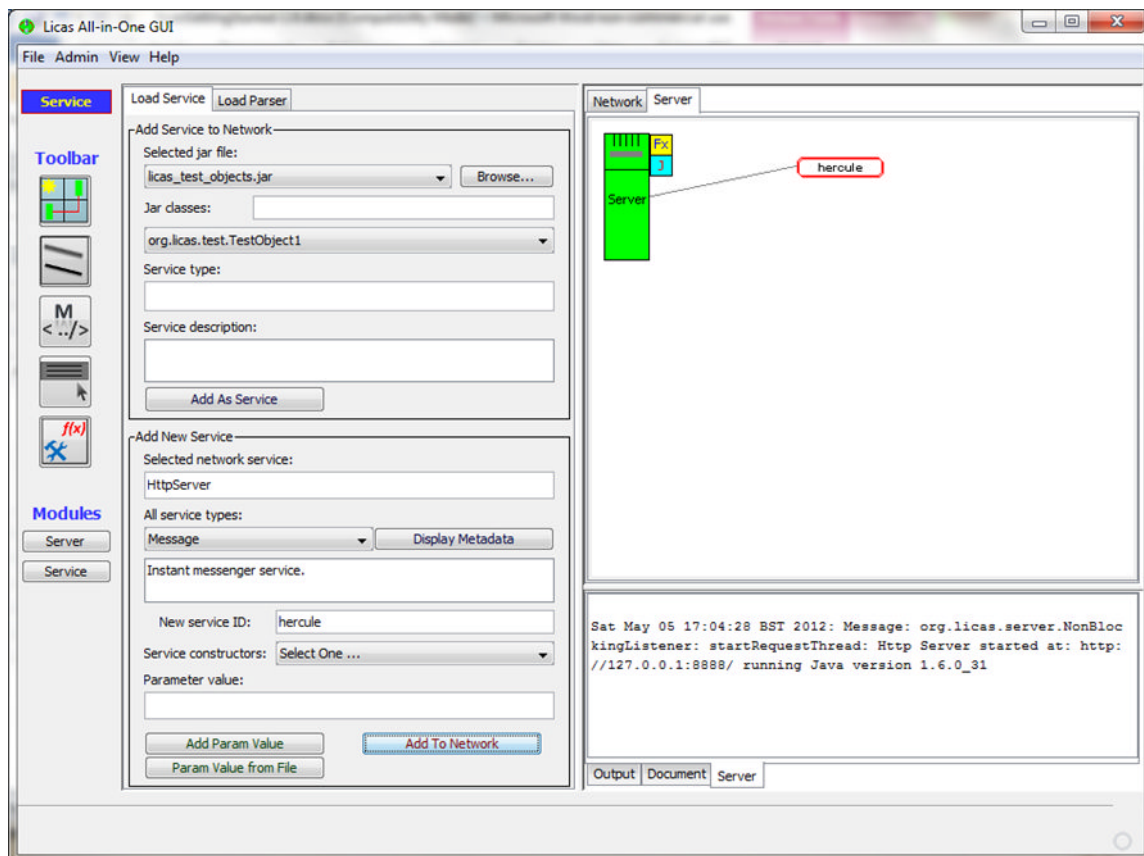


Figure 2. Service panel with default message service loaded.

If you hover the mouse over the graphic of the service and right-click it, you should open up a pop-up menu with different options. One of these options is called `GUI` and this option can allow a new GUI interface to be associated with the specified service, where the default classes provide a GUI interface for the message service. Before you can do this, you need to load in a `ServiceFactory` object, which is described in the next section. Note that the server shows if a service factory is running with an additional square, with `Fx` written in it. There is also a jar factory that can be started to process (remote) requests for jar files from the server. If running then this is shown by another square, with the letter `J` written in it.

3.3 Add a ServiceFactory Object

A ServiceFactory object controls what default services can be loaded and what GUI interfaces are associated with which service types. This is completely configurable and so you can also write your own one to change or add new operations. The default classes provide a GUI interface for the message service, which you can use without any other programming. One of the features of the licas package is the fact that you can load in classes for jar files that are not part of the compiled package. They could have been built and compiled separately, and the `licas_services.jar` jar file is this sort of package. It does extend the base licas package, but it has been built separately from the GUI and is therefore plugged in more like a new module. To demonstrate that this remote loading actually works, you need to perform this action to load in the default `ServiceFactory` object. If you click on the `Load service factory` toolbar button (the bottom toolbar button), then another form should open, as shown in Figure 3.

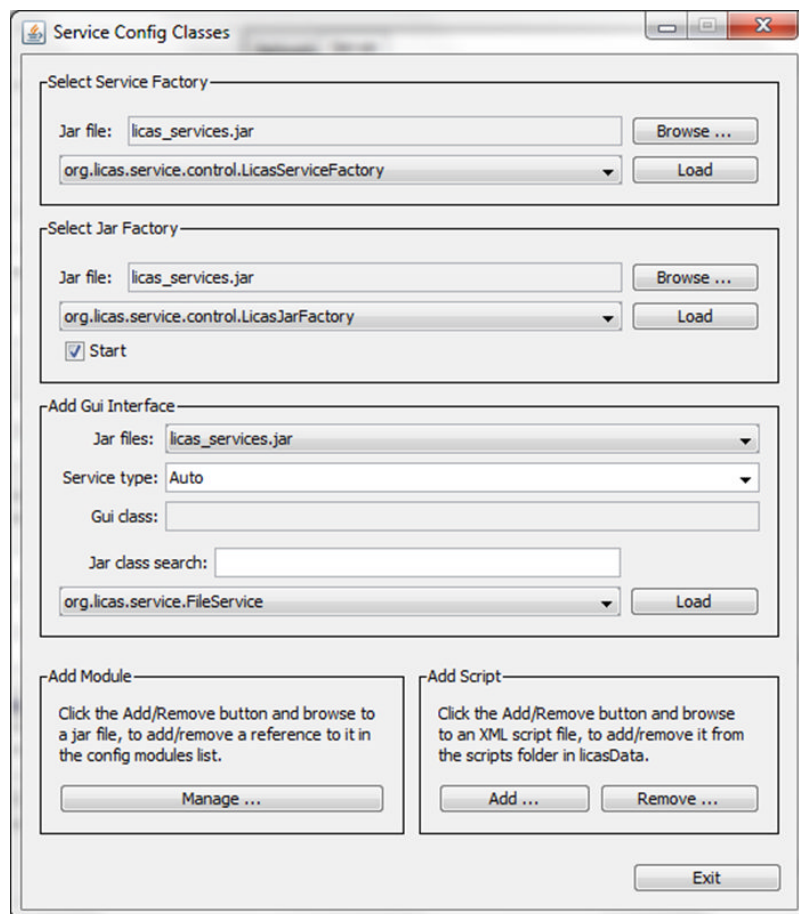


Figure 3. GUI Configuration form for selecting the service factory object.

In this form, click the `Browse` button and browse to the default `services` folder in the default `licasData` folder that has been added as part of the installation. This can be found in your root user directory. In the `services` folder there should be the `licas_services.jar` jar file. Select this file to open it, as shown in the figure. When you open this file, the classes are parsed and checked to determine if they extend the default abstract `ServiceFactory` object. Any that do are listed in the combo box. Only one class should be listed from the selected package, called `LicasServiceFactory`. If you then click on the `Load` button, the server view should change to show that a service factory has been added. Refresh the view to make sure that it has been added. You are also given the option to make this the default factory and to save the setting permanently to the config file again. You should select 'yes' to both of these. You can perform exactly the same operation to load in a `JarFactory` object, where a default one is again provided. This allows for jar files to be sent and saved on remote computers, to be automatically used to load a service onto the remote server. More details about this are given in the other documentation.

The GUI interface class types that the service factory uses are also stored and read from a file. In the root `licasData` folder, there is a file called 'licas_config.xml'. This is the configuration file for the `licas` GUI and also stores service types with their associated interface class names. The default package, for example, contains the following association:

```
<Service_Type Name="Message">
  <Gui_Classname>org.licas.service.gui.MessageServiceJFrame</Gui_Classname>
  <File>C:\Users\Kieran\licasData\services\licas_services.jar</File>
</Service_Type>
```

This is probably only of interest if you intend to write your own services, but it defines that a 'Message' type service can have a GUI interface of type 'MessageServiceJFrame'. The class will be loaded from the specified jar file. The Gui configuration form can also be used to add other service interfaces or load in other modules by default, where the 'licasGui' guide describes this in more detail.

3.4 Open a Service GUI Interface

After a service factory has been added, that object will determine what services have any GUI interfaces associated with them. If you right-click on your message service in the server graphic (hercule in this case), a pop-up menu should show a GUI option. If you now click that option, if there is a GUI associated with the service type, it will then open. If there is not, then you should receive an error message. For the message service, a message service GUI should open, as shown in Figure 4. The message service form will initially open with an empty address book. You can add addresses by clicking on the `Add Address` button. The

addresses are saved in the default 'licasData' folder, as an XML file. The address book is also shown in Figure 4.

To use the system, each service needs to know the name, location and password of the service that it wants to call. The passwords include a password for the server the service runs on and for the service itself. This is the basic security provided by the system, as otherwise any service is open to be invoked. However, if you do not enter or change the password settings, then they default to 'anon' and so the address book opens with these default values as well. The user name is the name of the service. This is 'hercule' in the local case and for the message service to be called, it is 'duke'. The server address is the IP address of the remote server. This requires the ip number and also the port number. Note that if the connection is to a computer outside of the internal network, your firewall might need to be configured to allow connections through it on a particular port number. Your system administrator might need to help with this. After the address details have been added, you can click the `Add` button to save them permanently and also write them to the address book file. You can then click the `Exit` button to close the address book form. The next section describes how to use this application to pass messages to another message service, running at some other location.

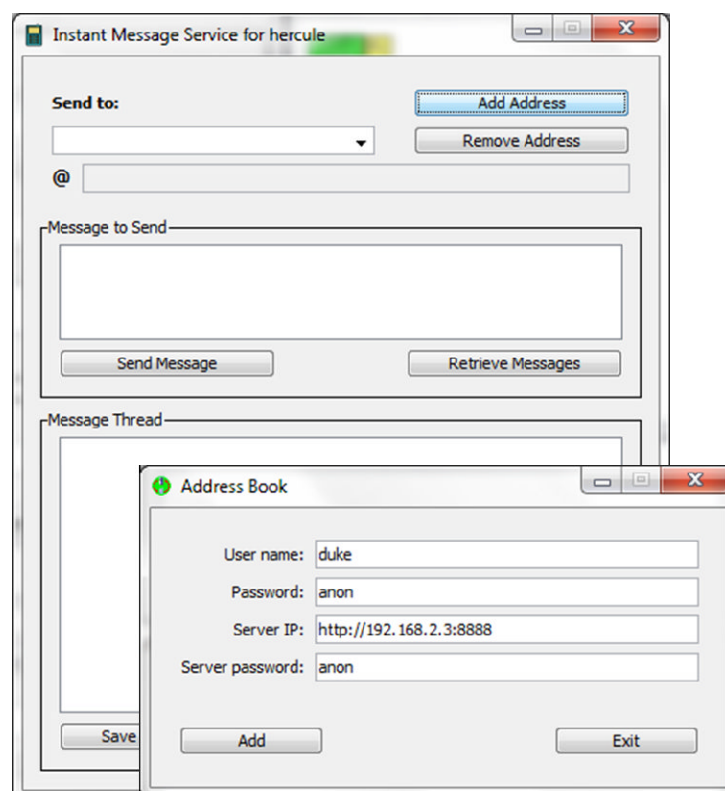


Figure 4. New message service window, also with address book form.

3.5 Using the Message Service GUI

If you perform the previous actions on two different computers, you will setup the licas application and load in a message service with its own GUI interface. You can then add the correct address to call for each application and you will have an instant message service. Figure 5 shows two instant message services started on different computers. It also shows a conversation between these two services. To make a call, you select the address of the person you want to send a message to. You then write the message in the Message to Send text area and click the Send Message button. The application runs a thread that periodically checks for new messages.

While this is still only a demo application, this will check if new messages for the specified persons have been received and display them in the Message Thread text area. If you are lucky, it will also display a balloon tip each time a new message is received. You can also select a new address and click the Retrieve Messages button to automatically update a conversation. The other options allow you to remove a conversation or save it to a file.

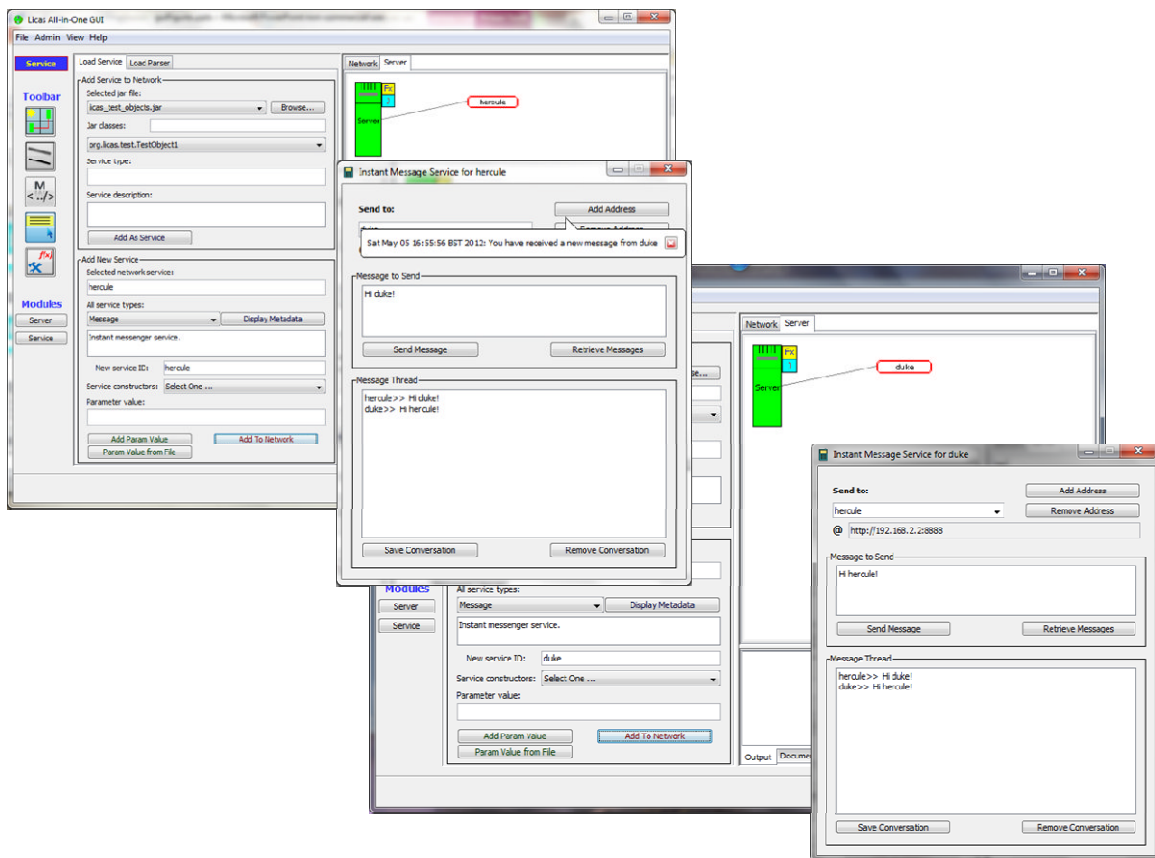


Figure 5. Instant message services started on two different computers.

4 File Service

The file service also has an interface, shown in Figure 6. You can load a file service onto a server and then open the GUI interface, in the same way as for the message service. The philosophy is for a service to communicate with another service. The GUI interface is used only to make method invocation easier. Because the All-in-One GUI can open a view on any server – remote or local, it would be possible to interact with a remote service directly through an interface. To keep with the basic design philosophy however, this implementation causes the interface to call the local service, which then calls and interacts with the remote one, etc.

This version can also be used as admin on the remote site. You can `Browse` to set the local directory root folder. Only folders and files underneath that one will then be accessible. The service will not backtrack to any parent of the specified file path

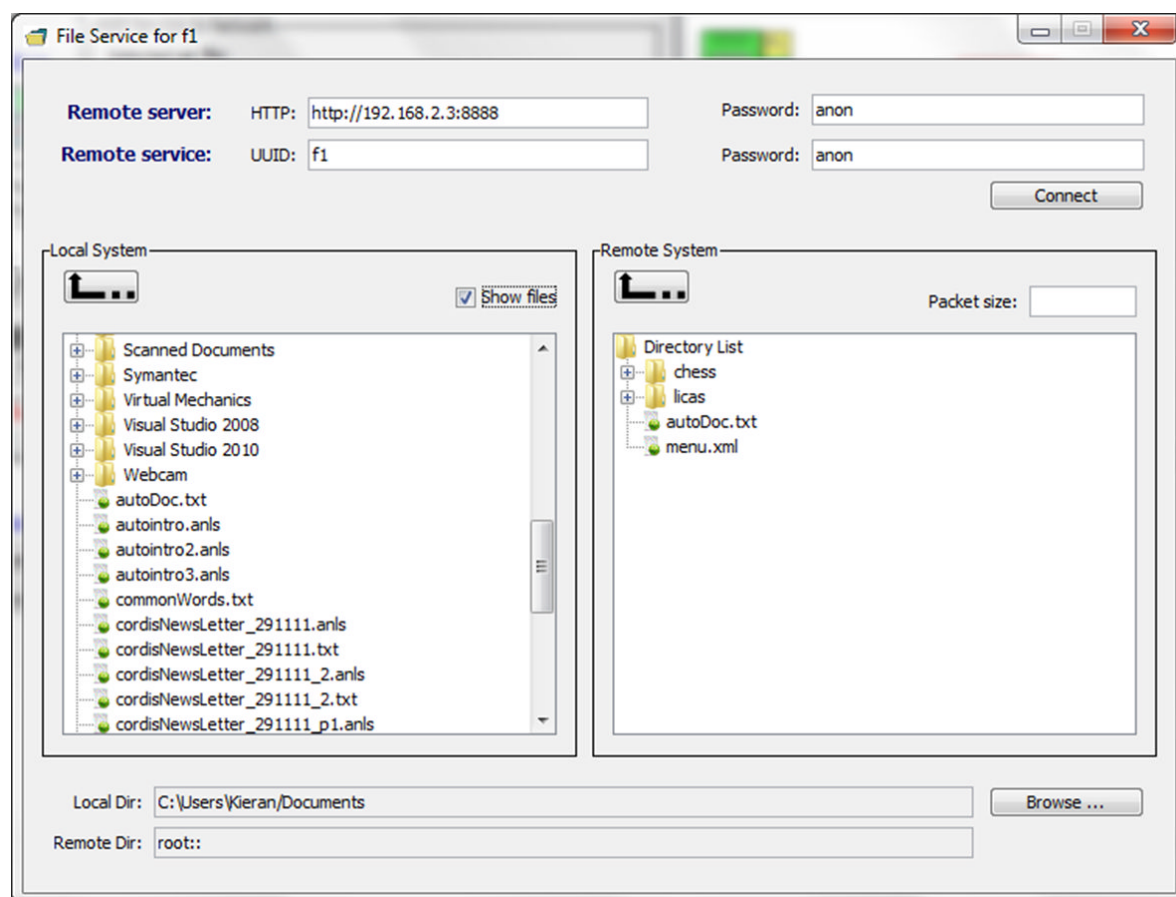


Figure 6. File Service Interface.

To copy a file, you enter the details of the remote file service and server that you want to invoke. You then click the `Connect` button to retrieve the remote directory root folder details. Clicking on a folder will move to that folder and the new path should be displayed at the bottom of the GUI. To copy a file, drag it from the remote directory tree to anywhere in the local directory tree area. It should then be copied to the specified local folder.

The large arrow button at the top moves the path back to the parent of the current path, but not beyond the specified root folders. The local directory files can be shown or hidden, as the local directory is not copied from, only copied to. The file transfer can also be done in stages, using a smaller packet size, if that is specified. See the user guide for more details about that.

5 Running a Server

To run a server, simply run the batch or script file that is included in the server folder. The J2SE version can be found in the `server` folder, while the mobile version can be found in the `server_mobile` folder. If you run the mobile version, you will open a small window on your computer. This window would represent the screen of a mobile phone if run in an emulator, but without that it appears as a stand-alone window. To start the server you then press the `Start` button and to stop it you press the `Shutdown` button. The server now runs on two adjacent ports. One port handles synchronous messages while the other handles asynchronous messages. You only need to enter one `port` value, for the synchronous handler, and the asynchronous handler will automatically be started on the `port+1` value. If the ports are not available then the server should probably register an error, but you should select a port value that allows the next one to also be available.

There is a log file, which is initialised with a configuration file that can be found in the configuration folder and is called 'logger.config'. You can change this to change the log file settings to write to different streams, etc. The log file used is the `jlog2` package that is also provided as open source. It can also be downloaded from the Web site and is provided with full documentation.

The server will naturally throw a lot of exceptions during its operation. This could be for any number of reasons including not finding the appropriate method on a particular object. These exceptions will all be displayed, but they can be ignored. If the system appears to be working properly then you do not need to worry about any exceptions that are thrown.

6 Running a Client

The server can be invoked either through a java-based GUI application, or through an HTML page in a browser, for example. The HTML option should make the server more accessible to mobile devices. A java-based GUI application is provided with the download that allows you to perform or test operations over data or services that are loaded onto a server. To run the client, simply double click the `runGui` batch or script file that is provided in the client folder. The client can only run on a PC or Laptop as it is built using J2SE, but it can be used to test either of the server versions (J2SE or J2ME). See the other documents for details on what the client application can be used for (`licasGui`) and in section 3 of this document. You do not have to use the gui to use the licas server however, and so it is possible to write a much simpler application for your own needs and run a licas server independently of the other packages.

7 Restful-Style Invocation of the Server

Because a java-based GUI application can be relatively heavyweight for a mobile device, you can also invoke a server through an HTML-based web page, or something similar. This should make writing client applications much easier. The server is still purely java-based however and so to use it, or add you own services to it, would require a java-based program. The provided GUI could be used for that.

As the licas server is a web server, it is also possible to make a method call or invocation through a string-based message that has a similar format to a restful web service. You can therefore create a web page or other, using HTML for example, and then use the HTTP POST method to send a single string-based message to the server. The server can receive this message and convert it into a `MethodInfo` object for further processing. The string format is similar to a restful web service and therefore is less complex than a parsed `MethodInfo` object. It is essentially a list of parameter names and values. The type checking is mostly missing, but some conversion to simple types is attempted by the licas server itself. There are also some standard parameter names that define specific parts of the method call. These are as follows:

- **service:** this is used to define the service to call, with the value being the service's uuid.
- **server_password:** this is the password for the server itself.
- **service_password:** this is the password for the service.
- **method_name:** this is the name of the method to invoke on the service.
- **return_type:** this is the type of object that should be returned and is optional. If this parameter is left out, a check on the return type is not made.

- The method parameters will then have their own id and value.

The message string separates each name-value pair with the '&' symbol and each name and value with the '=' symbol. The following message part:

```
service=service1&server_password=sp1&service_password=anon&method_name=getData&param1=abc&param2=def
```

Would therefore look for a service called `service1`, using `sp1` as the password to access the server and `anon` as the password to access the service itself. It would then try to invoke a method called `getData` on the service that would have two parameters, which are assigned the values `abc` and `def` respectively. Parameter type checking is currently omitted, but the process will try to parse the assigned values into the types of the first method that matches the specification (`getData` with 2 parameters). To make a call you also need to know the URL of the server, where you would write something like the following in javascript:

```
url = "http://127.0.0.1:8888";  
message = "service=service1&server_password=sp1&service_password=anon&method_name=getData&param1=abc&param2=def";  
request = getHTTPXMLRequest();  
request.open("POST", url, true);  
request.onreadystatechange = methodReply;  
request.send(message);
```

The use of AJAX with asynchronous message calling would be the preferred way to build a client application. So you can create an HTML page or similar and use AJAX with javascript to get the request object to make the call with. The reply is then a return string that can also be retrieved from the request object.

8 Running the Problem Solver

A problem solver package has been included as another separate package. You can use this along with a test configuration script to run problem solving tests over the system. This can also be linked to an existing network. The `licasHyperHeuristic` document describes in detail how to use the problem solver. The jar file is stored in the solver folder and the `runTests` batch or script file will run a test as specified by the configuration. The GUI now also contains a problem solving panel. It still tries to run a script, but the script file can be configured through the GUI instead of manually.

9 Additional Platform Requirements

The licas software has only one version of the source code. This however has been adapted to work with either a Java J2SE or a J2ME CDC environment. These can both be found on the Sun Java web site. Because of this, some additional jars are required for either environment. These are as follows:

If developing under the CDC 1.0, you will require the additional jsr jar of `jsr75_1.0.jar`, `jsr172-1.0.jar` and possibly the `cdc_1.0` jar itself, if compiling under J2SE. The new logging package `jlog2.jar` is also included. This is a new package that has been written and released as an open source project itself. The mobile version also requires a new lightweight parser and the one selected was the `nanoxml` parser. This has been wrapped by the classes in the `licas_xml` jar file to make it compatible with the licas system. Then also required are the `HttpCore` classes from apache for the Http server. Details of these additional jar file can also be found in the readme text document.

10 User Guides

There are a number of documents provided with the licas system. At the moment they are being kept as separate documents, where the content is as follows:

- The `licasGettingStarted` guide (this document) describes the different installation options.
- The `licasArchitecture` document gives a general overview of the whole system structure. This describes how the service components are connected in a network and how to communicate with or link to them.
- The `licasGui` document describes the default GUI application that is provided with the system. It is not open source, but is provided for free, for testing or non-commercial use. The document describes the different panels in the GUI and how to use them. You do not need all of these to run a network or test the system, where the feature set is now quite rich.
- The `licasService` document describes how you can add your own services or behaviours to the system. There is also a detailed description of how to add an interface to a service, turning it into a Cloud or SOA application.

- The `licasAdminGuide` document describes the metadata structure for describing the services. This includes the internal metadata describing the service's functionality, or metadata for service level agreements and security. You only need to worry about the metadata if you intend to test the security features or if you are interested in testing semantic processing. You do not need any additional metadata to create a network and invoke the services on it.
- The `licasUserGuide` document describes in detail some of the programming methods for writing code using the system. You will need to read this before you can do any programming. It is relatively easy to use the system once you have grasped the general idea of how to add a service and then communicate with it.
- The `licasHyperHeuristic` document describes how to use the problem solving framework that has been written on-top of the base `licas` packages.