



CERN-IT

# LIGHT

---

## How to get started

User's documentation

Version: 1.1.0  
Status: Final  
ID: IPT-LIGHT-UHGS  
Date: 05 June 2000  
Created: 12 April 2000  
Authors: D.Borillo



European Laboratory for Particle Physics  
Laboratoire Européen pour la Physique des Particules  
CH-1211 Genève 23 - Suisse

## Document Control Sheet

**Table 1** Document Control Sheet

<b>Document</b>	<b>Title:</b>	LIGHT How to get started		
	<b>Version:</b>	1.1	<b>ID:</b>	IPT-LIGHT-UHGS
	<b>Issue:</b>	0	<b>Status:</b>	Final
			<b>Created:</b>	21 October 1998
			<b>Date:</b>	05 June 2000
	<b>Stored at:</b>	\$CVSROOT/light2/doc/private/infrastructure		
<b>Tool</b>	<b>Name:</b>	Adobe FrameMaker	<b>Version:</b>	5.5
	<b>Template:</b>	Software Documentation Templates	<b>Version:</b>	xxx
<b>Authorship</b>	<b>Coordinator:</b>	A.Aimar		
	<b>Written by:</b>	D.Borillo		
	<b>Reviewed by:</b>			
	<b>Approved by:</b>			

This document has been prepared using the Software Documentation Layout Templates that have been prepared by the IPT Group (Information, Process and Technology), IT Division, CERN (The European Laboratory for Particle Physics). For more information please contact [docsys@ptsun00.cern.ch](mailto:docsys@ptsun00.cern.ch).

## Document Status Sheet

**Table 2** Document Status Sheet

<b>Title:</b> LIGHT How to get started			
<b>ID:</b> IPT-LIGHT-UHGS			
<b>Version</b>	<b>Issue</b>	<b>Date</b>	<b>Reason for change</b>
1.1	0		Reviewed by A.Kodahbandeh



# Table of Contents

<b>Document Control Sheet.</b>	<b>ii</b>
<b>Document Status Sheet</b>	<b>iii</b>
<b>Table of Contents</b>	<b>v</b>
<b>1 Overview</b>	<b>1</b>
<b>2 Publisher Interface</b>	<b>3</b>
2.1 The GUI	.3
2.2 The Interactive Console	.4
2.3 The Batch Console	.5
2.4 Installation procedure for the publisher interface	.5
2.4.1 Directory creation and environment variable settings	.5
2.4.2 Java Development Kit and Run-time environment	.6
2.4.3 Installing and running the Plug-in server (Apache)	.6
2.4.4 Installing the Apache server	.6
2.4.5 Configuring and running the Apache server	.7
2.4.6 Setting the environment for the publisher system	.7
2.4.7 Preparation procedure	.7
2.5 Run procedure	.7
<b>3 LIGHT servers</b>	<b>9</b>
3.1 The registration server	.9
3.2 Installing and running the registration server	.9
3.2.1 Preparation procedure	.9
3.2.2 Run procedure on MS-Windows	.9
3.2.3 Stop procedure on MS-Windows	10
3.2.4 Run procedure on Unix	10
3.2.5 Stop procedure on Unix	10
3.3 The data server	10
3.4 Installing and running the data server	11
3.4.1 Preparation procedure	11
3.4.2 Run procedure on MS-Windows	11
3.4.3 Stop procedure on MS-Windows	11
3.4.4 Run procedure on Unix	11
3.4.5 Stop procedure on Unix	12
<b>4 Example</b>	<b>13</b>
4.1 Installation procedures	13
4.2 Setting the Java policy file	13
4.3 Run the example step by step	14

4.4 Understanding the example . . . . . 15

**A LIGHT publisher API . . . . . 17**



# 1 Overview

This document is a guide for the LIGHT (Logical Information Global HyperText) user defined as “How to get started”. It contains information on how to install and run the different components of LIGHT which can be found on the web:

`http://light.cern.ch`

It is supposed that all the needed components have been download from the web and stored locally.

This document is formed by four chapters:

1. General structure of the document.
2. Publisher Interface. Description about how to get started with the publisher side of the system.
3. LIGHT servers. Description about how to get started with the servers (both registration and data servers).
4. Example. Some steps to follow in order to run the example provided on our web site.





## 2 Publisher Interface

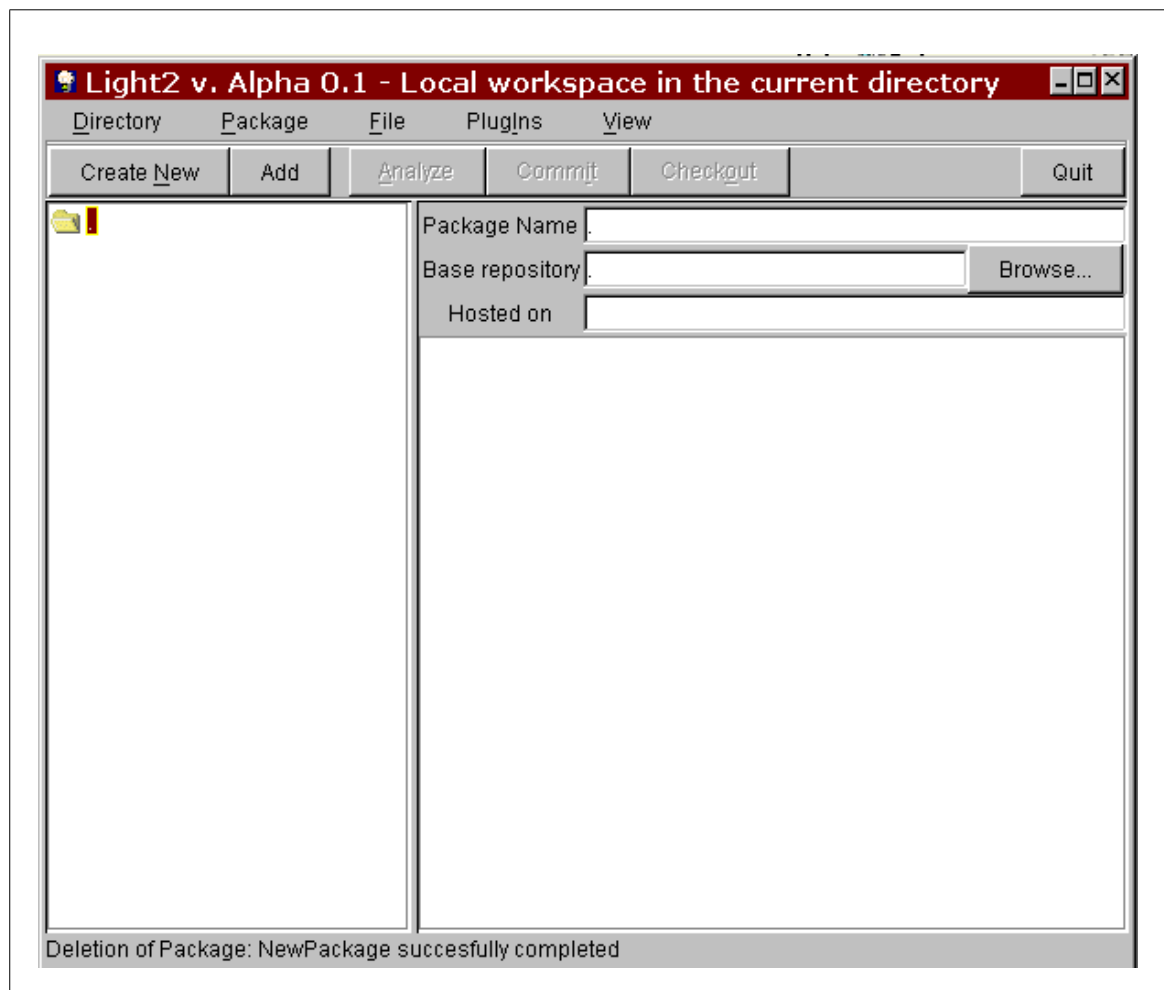
The publisher interface is the way through which you can produce LIGHT objects (and store them into LIGHT) by analyzing your documents.

It is a 100% pure Java stand alone application composed by three user interfaces:

- the Graphical User Interface (Section 2.1);
- the Interactive Console (Section 2.2);
- the Batch Console.

These interfaces are all communicating between them and they allow simply to have different types of access to the publisher API.

### 2.1 The GUI

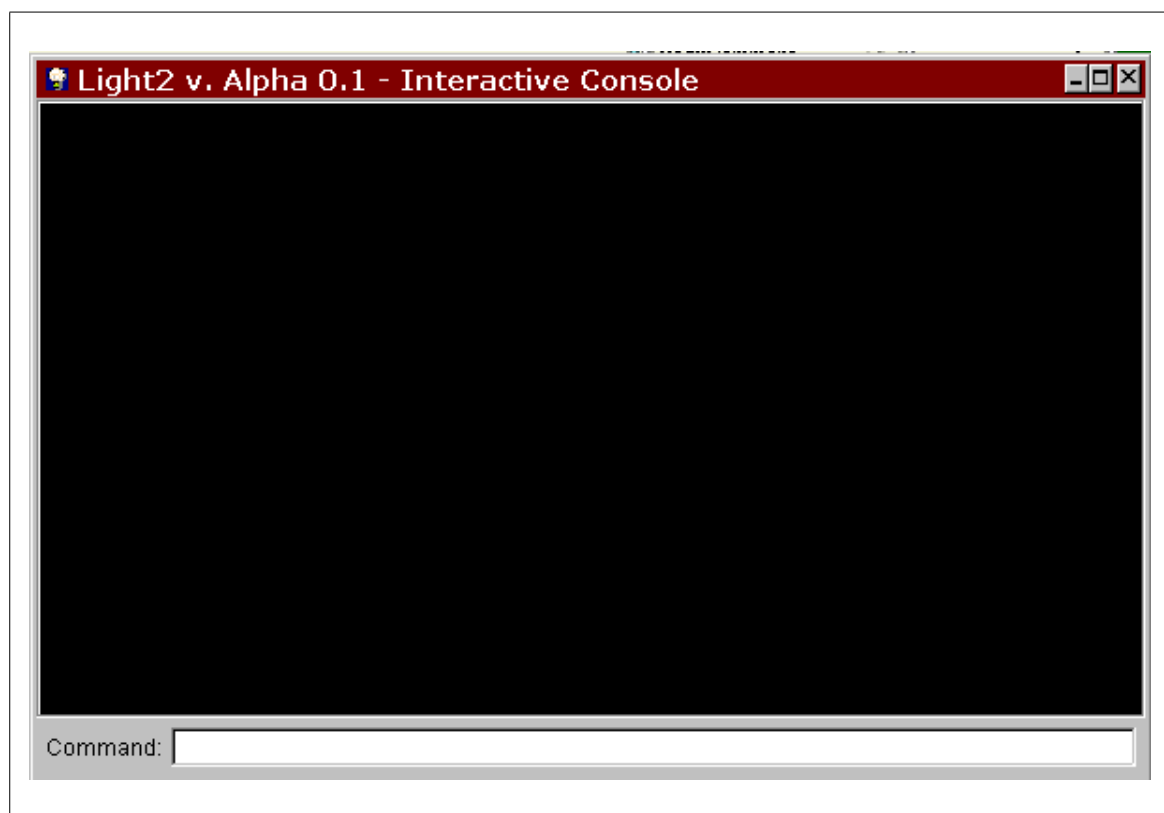


**Figure 1** A screen dump of the GUI

The GUI is the first window which appears when you start the LIGHT publisher interface. The API is accessed from here in a graphical way. Also the information about the packages created and their content are displayed graphically.

By opening at the same time the GUI and the Interactive Console (see Section 2.2) you can see which are the API commands that run performing the different types of graphical actions, since there is a direct correspondence between the action executed on the GUI and the interactive commands.

## 2.2 The Interactive Console



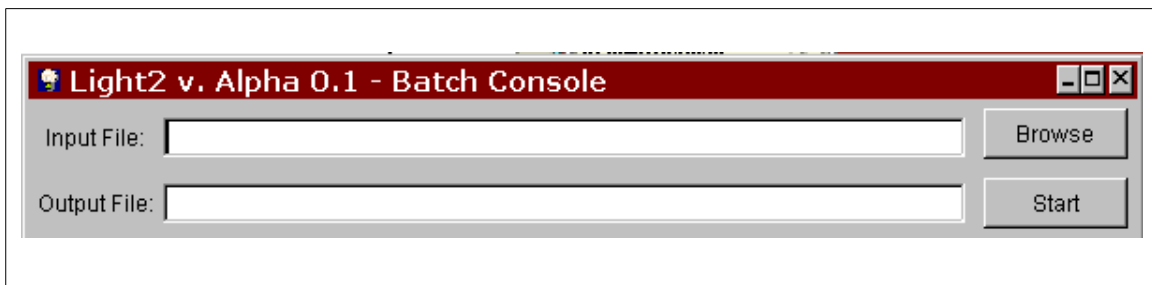
**Figure 2** A screen dump of the interactive console

Through this console the publisher API can be access by typing the commands (or short-cuts to the commands) in a shell style.

It is formed by a display component which shows messages deriving from the execution of the commands and a line input component, using which you can type and execute the different commands needed to perform the actions.

In order to run the interactive console you need to start the GUI (see Section 2.1) and from the menu bar select View -> Interactive Console.

## 2.3 The Batch Console



**Figure 3** Screen dump of the batch console

The batch console allows to run LIGHT Batch Files (with extension .lbf). These files are sequence of LIGHT commands which will be executed in batch mode. In the Input field you have to specify the location of the batch file you want to execute, while in the Output field a log file location will be specified.

In order to see an example of LIGHT Batch File you can refer to Section 4 or you can try to build your own sets of commands by referring to the command description in Appendix A.

## 2.4 Installation procedure for the publisher interface

This section describes how to install all the components needed in order to install and run the LIGHT publisher interface. All the modules and tools needed are referred from our web in the section Downloads.

### 2.4.1 Directory creation and environment variable settings

1. Create a local directory under which you will install all the LIGHT components (i.e. `C:\Light\install` on Windows or `/home/Light/install` on Unix).
2. Click on Start -> Settings -> Control Panel.
3. Double-click on System.
4. Choose the Environment tab.
5. On the Variable field type: `LIGHT_INSTALL`.
6. On the Value insert the base directory under which to install the LIGHT components (e.g. `C:\Light\install`)

On Unix systems (csh):

1. From the command line type: `setenv LIGHT_INSTALL = <your_home_dir>`  
(e.g. `setenv LIGHT_INSTALL = /home/Light/install`)

## 2.4.2 Java Development Kit and Run-time environment

LIGHT works with the Java 2 SDK, Standard Edition version 1.2 or more. If you do not have already install it you can download directly from the Javasoft web site following the link from our web: Downloads -> Tools & Libraries.

To test if the Java 2 SDK is already installed on your system, from a shell (MS-DOS or Unix) type:

1. `java -version`

In order to be able to run the light components correctly you need also to add the location of the Java bin directory to your path:

1. Click on Start -> Settings -> Control Panel.
2. Double-click on System.
3. Choose the Environment tab.
4. On the Variable field type: PATH.
5. On the Value insert the base directory under which you have installed Java bin Run-time environment (i.e. `<jre_bin_dir>;%Path%`)

On Unix systems (csh):

1. From the command line type: `setenv PATH = [jre_bin_dir]:$PATH`

## 2.4.3 Installing and running the Plug-in server (Apache)

The Apache server is used by LIGHT for serving Java classes grouped in jar files for browser and for the publisher system (as a RMI client) in order to allow remote class loading. It can be run on the same machine as data server is hosted since it serves the same jar files as used by data server, but also can be hosted on a separate machine. The CODEBASE property of each data server must point to the jar file containing classes which have to be loaded remotely on the Apache server (Light2analyzers.jar).

## 2.4.4 Installing the Apache server

You can download the Apache server from the link on the section Downloads -> Tools & Libraries of our web site.

Following this link you will find all the necessary information on how to install the server. From now on we will suppose that you have installed it in the directory `%LIGHT_INSTALL/Apache`.

## 2.4.5 Configuring and running the Apache server

1. Modify the file `%LIGHT_INSTALL%\Apache\conf\httpd.conf` setting the parameter:
  - `ServerName your_machine`
  - `DocumentRoot <Value of %INSTALL_LIGHT%/hserver>`
  - `<Directory <Value of %INSTALL_LIGHT%/hserver>>`
  - On Windows the default port value is 80.
  - On Unix machines without the root privileges you will not be able to run Apache on the default http port. Thus we propose to set the different port number: 2099
2. run the Apache server following the instruction provided with the tool.

## 2.4.6 Setting the environment for the publisher system

In order to install the publisher system, it is necessary to have stored locally the LIGHT publisher client tar file (`publisher.tar`) which can be download from the section Downloads -> Executables of our web site.

## 2.4.7 Preparation procedure

1. Untar the file `publisher.tar` into the installation directory referred by the variable `LIGHT_INSTALL` (this will create a sub-directory `%LIGHT_INSTALL%\publisher`).

## 2.5 Run procedure

1. Go to the publisher directory (`%LIGHT_INSTALL%\publisher`).
2. Run the file `run.bat` by double clicking on it (on MS-Windows systems) or run (on Unix systems).



## 3 LIGHT servers

There are two different kinds of servers in the LIGHT system which fulfil different issues: the registration server (see Section 3.1) and the data server (see Section 3.3). Both servers are written in 100% pure Java and in principle they run on every OS where a JVM can be installed. They have been tested on Windows NT 4.0, Solaris 2.6 and Linux Redhat 6.1.

### 3.1 The registration server

The registration server (referred as RS) is a broker which stores and addresses references to the different types of data available in the data servers, which means that it can handle references to analyzers, visualizers and LIGHT object in a transparent way from the point of view of the end-user. Both publishers and users of the system will always access the LIGHT data stored in the data servers through a specified RS.

The RS is composed by two main components (which must be run separately, see Section 3.2): one component is used to address data via the http protocol, way the other one is used to address data by the publisher system.

### 3.2 Installing and running the registration server

In order to install the RS, it is necessary to have stored locally the LIGHT registration server tar file (`regserver.tar`) which can be download from the section Downloads -> Executables of our web site.

#### 3.2.1 Preparation procedure

1. Untar the file `regserver.tar` into the installation directory referred by the variable `LIGHT_INSTALL` (this will create a sub-directory `%LIGHT_INSTALL%\rserver`).
2. Modify the files `%LIGHT_INSTALL%\rserver\settings.bat` (or settings on Solaris and Linux) as follow:
  - `LIGHT_RSHOST` and `LIGHT_RSPT` with your domain name and the port you want to use (by default the host is set to `%COMPUTERNAME%.cern.ch` and the port to 1099)

#### 3.2.2 Run procedure on MS-Windows

1. Go to the registration server directory (`%LIGHT_INSTALL%\rserver`).
2. Run the file `rmiregister.bat` by double clicking on it.

3. Run the batch file `run.bat` to start the RS (publisher application side).
4. Run the batch file `startserver.bat` to start the servlet application accessed using the http protocol.

### 3.2.3 Stop procedure on MS-Windows

1. Go to the registration server directory (`%LIGHT_INSTALL%\rserver`).
2. Run the `stopserver.bat` file to shutdown the servlet runner.
3. Terminate the `run.bat` script by closing its window.
4. Terminate the `rmiregister` by closing its window.

### 3.2.4 Run procedure on Unix

1. Go to the registration server directory (`$LIGHT_INSTALL/rserver`).
2. Run the file `rmiregister &` in background.
3. Run the script `run &` to start the RS (publisher application side).
4. Run the script `startserver` to start the servlet application accessed using the http protocol.

### 3.2.5 Stop procedure on Unix

1. Go to the registration server directory (`$LIGHT_INSTALL/rserver`).
2. Run the script `stopserver` to shutdown the servlet runner.
3. kill the `run` script by pressing `Ctrl-C` (first you may need to make this process running in the foreground by typing `fg`).
4. kill the `rmiregister` by pressing `Ctrl-C` (first you may need to make this process running in the foreground by typing `fg`).

## 3.3 The data server

The data server (referred as DS) stores physically all the LIGHT data (analyzers, visualizers and LIGHT objects) which are then addressed by the RS (see Section 3.1). In principle there can be more DS on which different data are stored (i.e. you can set one DS for the analyzers, one for the visualizers and one for the LIGHT objects).



## 3.4 Installing and running the data server

In order to install the DS, it is necessary to have stored locally the LIGHT data server tar file (`dataserver.tar`) and the LIGHT plug-ins and http server add-on tar file (`httpserver.tar`) which can be download from the section Downloads -> Executables of our web site.

### 3.4.1 Preparation procedure

1. Untar the files `dataserver.tar` and `httpserver.tar` into the installation directory referred by the variable `LIGHT_INSTALL` (this will create the sub-directories `%LIGHT_INSTALL%\dserver` and `%LIGHT_INSTALL%\hserver`).
2. Modify the files `%LIGHT_INSTALL%\dserver\settings.bat` (or settings on Solaris and Linux) as follow:
  - `LIGHT_DSHOST` and `LIGHT_DSPOINT` with your domain name and the port you want to use (by default the host is set to `%COMPUTERNAME%.cern.ch` and the port to 2098).
  - `LIGHT_HSHOST`, `LIGHT_HSPORT` and `LIGHT_HSPATH` (for the plugin server) with your domain name, the port you want to use and the path to the plugin server (by default the host is set to `%COMPUTERNAME%.cern.ch`, the port to 80 and the path to `/`).

### 3.4.2 Run procedure on MS-Windows

1. Go to the data server directory (`%LIGHT_INSTALL%\dserver`).
2. Run the `rmiregister.bat` file.
3. Run the batch file `run.bat` to start the data server.

### 3.4.3 Stop procedure on MS-Windows

1. Terminate the shell running the `run.bat` file by closing its window.
2. Terminate the `rmiregister` by closing its window.

### 3.4.4 Run procedure on Unix

1. Go to the data server directory (`$LIGHT_INSTALL/dserver`).
2. Run the `rmiregister` & script in background.
3. Run the script `run` to start the data server.

### 3.4.5 Stop procedure on Unix

1. Kill the script `run` by pressing the `Ctrl-C`.
2. Kill the `rmiregister` by pressing `Ctrl-C` (first you may need to make this process running in the foreground by typing `fg`).

## 4 Example

This section describes how to install and run all the components needed to perform the test example provided from our web site in the section Downloads -> Examples.

The example we refer in this section is the first one you can download from the site and it is called FSM2 example (`fsm2example.tar`). It is a very simple C++ program correlated with several Booch static diagrams.

The purpose of this very simple example (FSM2) is to allow you to get familiar with LIGHT and also to verify that all the components you have installed can run correctly

### 4.1 Installation procedures

In order to run the example it is necessary to install correctly all the components of LIGHT (client/server).

Section 4.2 assumes that you have followed the installation procedures for:

- publisher interface (Section 2.4);
- registration server (Section 3.2);
- data server (Section 3.4).

### 4.2 Setting the Java policy file

Before you can browse documents stored on the LIGHT servers, you need to download and store your `java.policy` file:

1. Download the `.java.policy` file from our web site (section Downloads -> Executables).
2. Save the `.java.policy` file into your profile directory (i.e. `C:\WNT\Profiles\<your_username>\.java.policy`). On Unix place the file in the `$HOME` directory.

The `.java.policy` file provided on our web grants java applets loaded from any host the permission to connect with any host on the http or any non-system port number:

```
grant codeBase "http://*/-" {  
    permission java.net.SocketPermission "*:1024-65535", "connect, resolve";  
    permission java.net.SocketPermission "*:80", "connect, resolve";  
};
```

You can set up your own policy file with more restrictive policies using the `policytool` provided by Sun Microsystems with the JDK package. With this tool set entries according to the rules:

```
grant codeBase "http://your_http_server/-" {
    permission java.net.SocketPermission "registration_server:port", "connect,
resolve";
    permission java.net.SocketPermission "data_server1:port", "connect, resolve";
    permission java.net.SocketPermission "data_server2:port", "connect, resolve";
    ....
};
```

If necessary, repeat the `grant` entry for different machines, or set the `codeBase` to your domain with the statement:

```
"http://*.cern.ch/-"
```

Save the created file in your profile directory (i.e.

`C:\WNT\Profiles\<your_username>\.java.policy`) file (or in your `$HOME` directory in case of using the browser on Unix).

## 4.3 Run the example step by step

1. Download the file `fsm2example.tar` from the section Downloads -> Examples of our web site.
2. Untar the file `fsm2example.tar` into the directory `%LIGHT_INSTALL%` (this will create the sub-directories `%LIGHT_INSTALL%\examples\fsm2`).
3. Modify the script `%LIGHT_INSTALL%\examples\fsm2\fsm2.lbf` substituting the variables values as follow:
  - set `registrationServer` <registration server host> (see Section 3.2.1 to get the value)
  - set `dataServer` <data server host>:<data server port> (see Section 3.4.1 to get the values)
  - set `pluginServer` <plugin server host>:<plugin server port> (see Section 3.4.1 to get the values)
  - set `baseDir` <absolute path of `%INSTALL_LIGHT%\examples\fsm2`>
4. If it does not exist already create the directory `%INSTALL_LIGHT%\examples\fsm2\local_ws`.
5. Run all the servers (registration server, data server and plugin server, see Section 3).
6. Run the publisher GUI console (see Section 2.5).
7. From the GUI console select in the menu View -> Show Batch Console.
8. On the batch console use the Browse button to choose the file `%LIGHT_INSTALL%\examples\fsm2\fsm2.lbf`.
9. Click on the Start button.
10. After executing `fsm2.lbf` check the log file (`fsm2.log`) for possible errors.

11. If no error has been detected open a web browser window (Netscape 4.x or IE 5.x) and open the following URL:  

```
http://<registration server  
host>:8080/servlet/LIGHTRegistrationServer
```
12. Navigate through the code and the diagrams analyzed and now displayed using LIGHT.

## 4.4 Understanding the example

What have we done running the `fsm2.lbf` file? Which is the logic that LIGHT follows when publishing documents? This section gives an overview of the step performed by the script, for further details on the commands used please refer to the Appendix A.

The command `set` is used first to set the variables that then will be used by the script and all the scripts invoked by it.

Then we are setting the location of the registration server to use via the command `rsset` and the local workspace via the command `workspace`.

The visualizer plug-in `LPackageIndex2` is used to browse the index of packages (which is the first displayed when browsing). For this reason, we used the command `visualizerregister` passing `EMPTY` as parameters for the LIGHT object type, subtype and action, since this visualizer is used for all the packages independently from their content.

The command `include` is used to call sequentially the other scripts needed in the analysis process.

The first script invoked creates the packages `fsm.source` and `fsm.source.include` with the command `packagecreate`. Then it adds the files to be analyzed to the correct packages through the command `fa` (`file add`).

Once the packages are created and the files are set the script allows to configure the connectivity rules via the command `packageconfigure`. The packages can then be committed with the command `packagecommit`.

The only remaining step for this script is to register the plug-in analyzer `LCPPAnalyzer` (which will then be used to run the analysis) via the command `analyzerregister`.

The second script performs similar tasks for the Booch static diagrams: creating and configuring a package, adding the files, committing the packages and registering the needed analyzer (`LGIEAnalyzer`).

The only step that it does in addition is to import the packages between themselves with the command `importpackage`. Importing a package into another one means that when searching for LIGHT objects, LIGHT will look into a given package and if the needed objects can not be located there they will be searched in all the imported packages.

Finally the third script associate file extensions with the correct analyzer using the command `formatregister` and then it runs the analysis of the different packages with the command `analyzerrun`.

It also register all the needed visualizer plug-ins to display the LIGHT objects by type and to visualize indices of LIGHT objects (i.e. display LIGHT objects of type `cpp_class` subtype declaration).

The last step is to commit once again the analyzed packages including the LIGHT objects generated during the analysis process.

When browsing the first time the content referred by the selected registration server the `LPackageIndex2` visualizer will be used.

## A LIGHT publisher API

This appendix describes (in alphabetical order) all the commands accessible via the interactive console, their related short-cuts (in parenthesis), the syntax of each of them and of course a brief description of their behaviour.

### **analyzerregister (are)**

Syntax: `analyzerregister analyzer_name registration_server data_server`

Register, in a given *registration\_server*, an analyzer identified by *analyzer\_name* (i.e. `ch.cern.light.ipt.cpp.analyzers.LCPPAnalyzer`) and physically stored into a *data\_server*.

### **analyzerrun (aru)**

Syntax: `analyzerrun package_name [input_document_name]`

Run the appropriate analyzer (previously registered) on a given package identified by *package\_name*. Optionally it can be specified an input document: in this case the analysis will be limited to this file.

### **fileadd (fa)**

Syntax: `fileadd file_pattern package_name`

Add a file (or a set of files) specified by the *file\_pattern* to a given package (specified by *package\_name*).

### **fileremove (frm)**

Syntax: `fileremove file_pattern package_name`

Remove a file (or a set of files) specified by the *file\_pattern* from a given package (specified by *package\_name*).

### **formatregister (fr)**

Syntax: `formatregister analyzer_name registration_server extension, [extension]*`

Associate an analyzer (*analyzer\_name*) which is registered on a given *registration\_server* to a set of file extensions. This will allow the system to recognize files by type of extension and run the correct analyzer when the command `analyzerrun` is invoked.

### **help**

Syntax: `help [-f command_name]`

Display the list of all the available commands on the display window of the interactive console. The switch `-f` followed by a command display the syntax and a brief description of each command.

## **include**

Syntax: `include lbf_file`

Run a LIGHT batch file (lbf). Often used inside lbf files to invoke other lbf files.

## **packagecheckout (pch)**

Syntax: `packagecheckout package_name [registration_server base_directory]`

Check-out a package referred by *package\_name*. If the package is not already existing on the local work space also the *registration\_server* and the directory where to map the package (*base\_directory*) must be specified, otherwise the current package values are assumed.

## **packagecommit (pcom)**

Syntax: `packagecommit package_name [registration_server data_server]`

Check-in (commit) a local package referred by *package\_name*. If the package is committed for the first time it is necessary also to specify the *registration\_server* where the package will be registered and the *data\_server* where it will be physically stored. Otherwise the current package values are assumed.

## **packageconfigure (pcon)**

Syntax: `packageconfigure package_name configuration_file`

Configure a package referred by *package\_name* getting the configuration data from a local file specified by *configuration\_file* (with extension lcf).

## **packagecreate (pcr)**

Syntax: `packagecreate package_name base_directory`

Create locally an empty package giving it an identifier specified by *package\_name*. The *base\_directory* parameter is used to specify where the files that will be added to the package are mapped on the local system (e.g. they will be added specifying their local path with respect to the base directory).

## **packagedelete (pd)**

Syntax: `packagedelete package_name`

Delete a package referred by *package\_name* from the local work space. The package will not be removed from the data server where it has been committed.



### **packageimport (pi)**

Syntax: `packageimport package_name package_imported registration_server`

Import a package referred by *package\_imported* (and registered in the *registration\_server*) into the package specified by *package\_name*. This means that if, while searching for a given LIGHT object, it is not found inside the *package\_name*, the system will look also in all the package imported using this command.

### **packageimportremove (pir)**

Syntax: `packageimportremove package_name package_imported`

Remove a package (*package\_imported*) imported with the command `packageimport`, from the package referred by *package\_name*.

### **packagelist (plist)**

Syntax: `packagelist [-f] package_name`

List the package specified by *package\_name* and all its sub-packages. If the `-f` flag is specified, the files belonging to each package listed will also be displayed.

### **packagemap (pm)**

Syntax: `packagemap package_name new_base_directory`

Remap the package referred by *package\_name* on a new local repository specified by *new\_base\_directory*. This command is useful when the location of the files contained into the package has been moved.

### **packagerename (prn)**

Syntax: `packagerename old_package_name new_package_name`

Rename a package referred by *old\_package\_name* assigning to it a new name specified by *new\_package\_name*.

### **packagesave (save,ps)**

Syntax: `packagesave [. || package_name]`

Save a package referred by *package\_name* and all its sub-packages into the local work space. If `.` is specified it will save all the packages on the local work space.

### **quit (exit,q)**

Syntax: `quit`

Exit from the publisher system.

### **ranalyzerlist (ralist)**

Syntax: `ranalyzerlist registration_server`

List all the analyzers currently registered on a specified *registration\_server*.

### **registrationserverset (rsset)**

Syntax: `registrationserverset registration_server`

Set the *registration\_server* to be the default registration server in the current session.

### **rpackagelist (rplist)**

Syntax: `rpackagelist registration_server`

List all the packages currently registered on a specified *registration\_server*.

### **rvisualizerlist (rvlist)**

Syntax: `rvisualizerlist registration_server`

List all the visualizers currently registered on a specified *registration\_server*.

### **set (setenv)**

Syntax: `set variable_name variable_value`

Allow to set variables that can be reused in other commands. The variables are identified by *variable\_name* and they can have a value specified in *variable\_value*. When using a variable this is access by its name with the syntax `%variable_name%`.

### **visualizerregister (vr)**

Syntax: `visualizerregister registration_server objtype objsubtype  
action visualizer_name data_server`

Register a visualizer, referred by *visualizer\_name* and located physically in *data\_server*, into a registration server specified by *registration\_server*. This visualizer will be used to display LIGHT objects of type *objtype* and subtype *objsubtype*. An *action* can be also specified to target the behaviour of the visualizer itself (i.e. *action* = index to visualize the index of the LIGHT objects searched).

### **workspaceget (wg)**

Syntax: `workspaceget`

Return to the display the actual publishing work space.

### **workspaceset (ws)**

Syntax: `workspaceset directory_name`

Set the publishing work space to the directory *directory\_name*. This means that all the LIGHT data produced locally will be stored in this directory.