User Manual

Document Name:	**Connecting Remotely Operable Labs to the Cyberlab Network - Introduction**
Document Date:	**2003-06-21**
Document Author/s:	**Tor Ivar Eikaas**
Document approved by:	
Document type:	**User manual**
Deliverable Number / Report number:	**UM-013**
Version:	**1.0**
Status:	**Final**
Classification:	**Confidential**
Document Reference Number:	**UM-013-01**
File:	**UM-013-ConnectingLabs-Intro-v1_0.doc**

TABLE OF CONTENTS

Tables

Figures

Summary

This document gives a brief introduction of the SW and methodology used for connecting remotely operable laboratories to the Cyberlab Network via the Cyberlab Communication Server and the Provider’s Lab Server. A special focus is on the methods, SW and interfaces for the communication between the Cyberlab Communication Server and the Lab Server.

This document is not meant to be a complete reference and description of the SW used for the Cyberlab Portal and Cyberlab Communication Server, but it is rather an overview giving the main functionality.

Further details and example code is available from Cyberlab.

1 Definitions

AA	Application Adapter
CCI	Cyberlab Communication Interface.
CCS	See Cyberlab Communication Server.
Customer	A Cyberlab User registered and accepted by Cyberlab as a Customer.
Cyberlab Admin	A specific Cyberlab User from Cyberlab.Org given Cyberlab Admin rights.
Cyberlab DB	The Cyberlab database. Located on a Cyberlab computer. May be split in several physical databases.
Cyberlab Communication Server (CCS)	The SW running at the Cyberlab's site providing the necessary functionality for interfacing the Lab Server via the Cyberlab – Provider Communication and the Cyberlab DB. Previously called Cyberlab Server.
ESP	Experiment Service Provider. This is the organization (in this case Cyberlab) responsible for the services offered both to Customers and Providers.
Experiment	A predefined exercise defined for performing some operation on a Laboratory.
Experiment Server	See Lab Server
Guest	A non-registered person accessing the Cyberlab Web. A Guest has only limited access to the site's functionality.
GUI	Graphical User Interface. Often also called HCI.
Lab / Laboratory	The whole setup of the physical apparatus, instrumentation and SW needed for a laboratory.
Lab User	A Cyberlab User accepted for access to one (or more) of the Labs.
Lab Unit	A Time Slot or a single access to a Lab. This is the shortest possible duration when booking access to a Lab.
Lab Server	The SW running on the Providers' site providing the necessary functionality for interfacing the Lab and the Cyberlab Server. In addition, the Lab Server may provide functionality for the communication with the Lab User Client SW when operating the Lab.
Provider	An organization owning a Lab.
Provider Admin	A Cyberlab User from the Provider organization given Provider Admin rights.
TA	Transportation Adapter
User / Cyberlab User	A person registered as a Cyberlab User.
Username	Identifies the person accessing the Cyberlab system (independent of his role as Cyberlab Admin, Lab User, Provider Admin etc).

Part I - System overview

2 General system overview of the Cyberlab System

The Cyberlab system is based on a Business Model where the three different actors communicate with each other from separate computers via Internet as illustrated in Figure 1 below.

The three actors are:

- *The Experiment Service Provider (ESP).*
  This is represented by the organising company (Cyberlab) who provides all the necessary services and access to the laboratories in order to have the system in operation.
- *The Providers*
  This is the laboratory owners with their individual physical lab apparatus.
- *The Customers*
  This is individuals or organisations acting as customers an performing experimentation on the remotely operable laboratories.

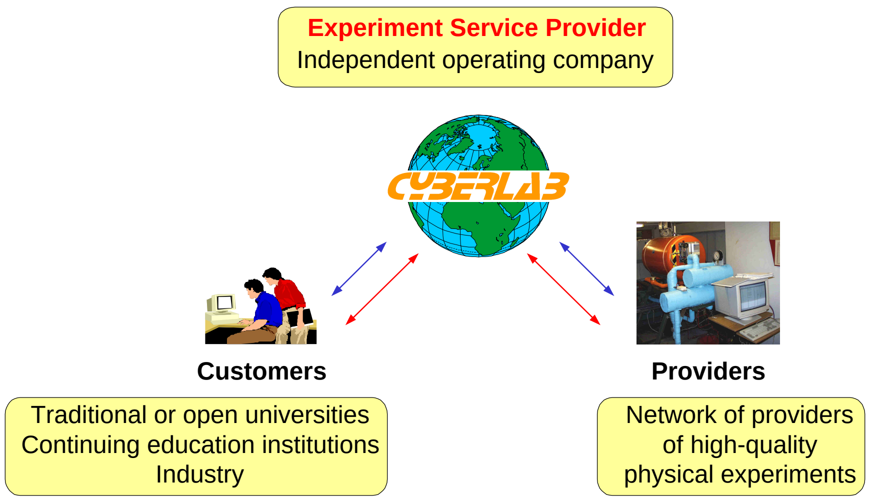

Experiment Service Provider
Independent operating company
CYBERLAB
Customers
Traditional or open universities
Continuing education institutions
Industry
Providers
Network of providers
of high-quality
physical experiments

Figure 1 Schematic view of the ESP business model

2.1 Main Cyberlab Network Services

The main services offered via the Cyberlab Network can be summarised as:

- The Lab portal, eCommerce and on-line 24/7 services:
  - Web-pages for Lab presentation, promotion etc.
  - Quality assurance.
  - Lab availability, regularity.
  - Booking, invoicing, reporting, etc.
  - Access control, usage supervision.
  - Lab supervision.
  - Support services, reporting, etc.
  - Handling of disputes
- Lab interface sw
  - Specifications and sw supporting a wide range of lab server sw solutions.
- Support and maintenance services for new and existing lab. Providers.

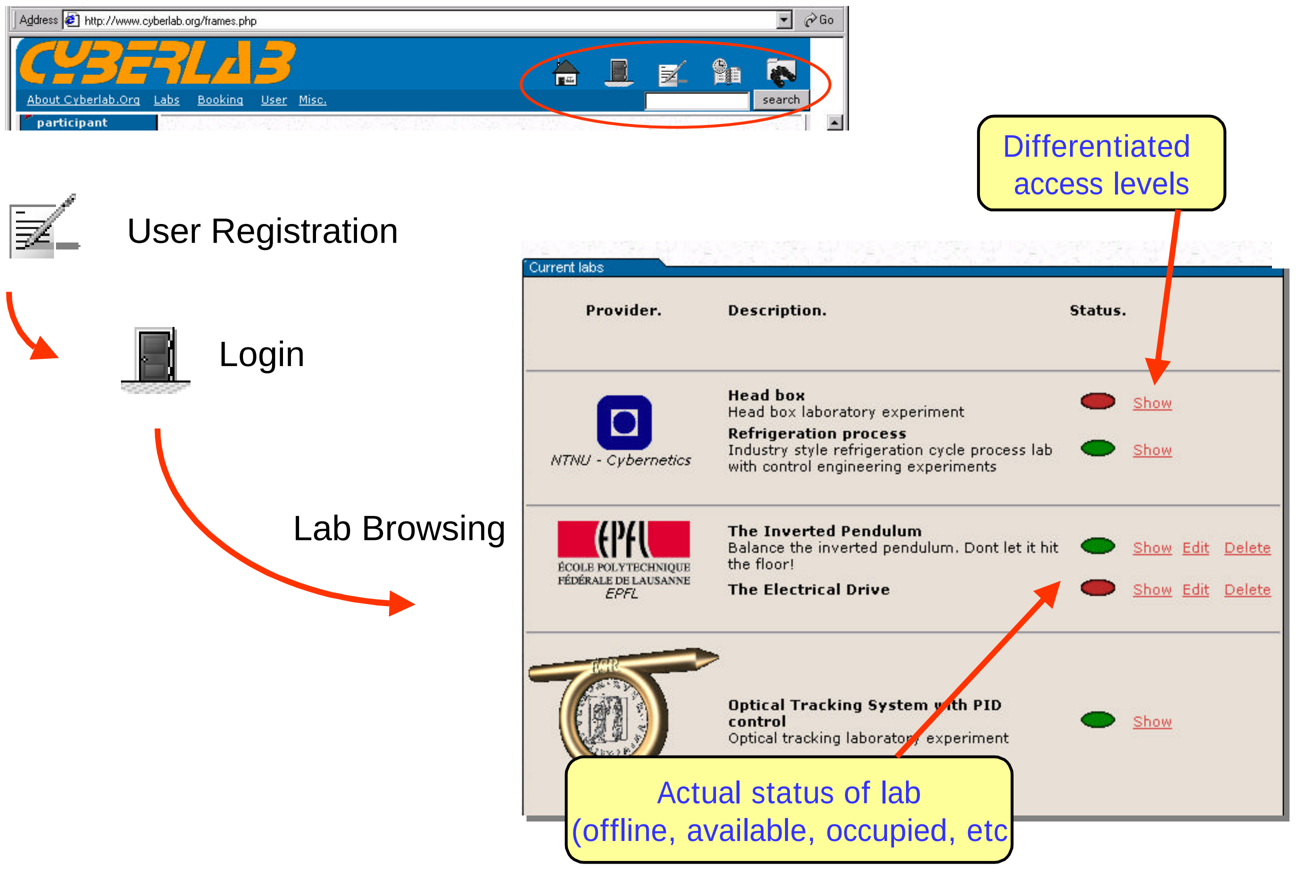
2.2 The Cyberlab Portal

The Cyberlab Portal provides differentiated user access. The main user categories are:

- Visitors
- Users and Customers
- Provider Admins
- Cyberlab Admin

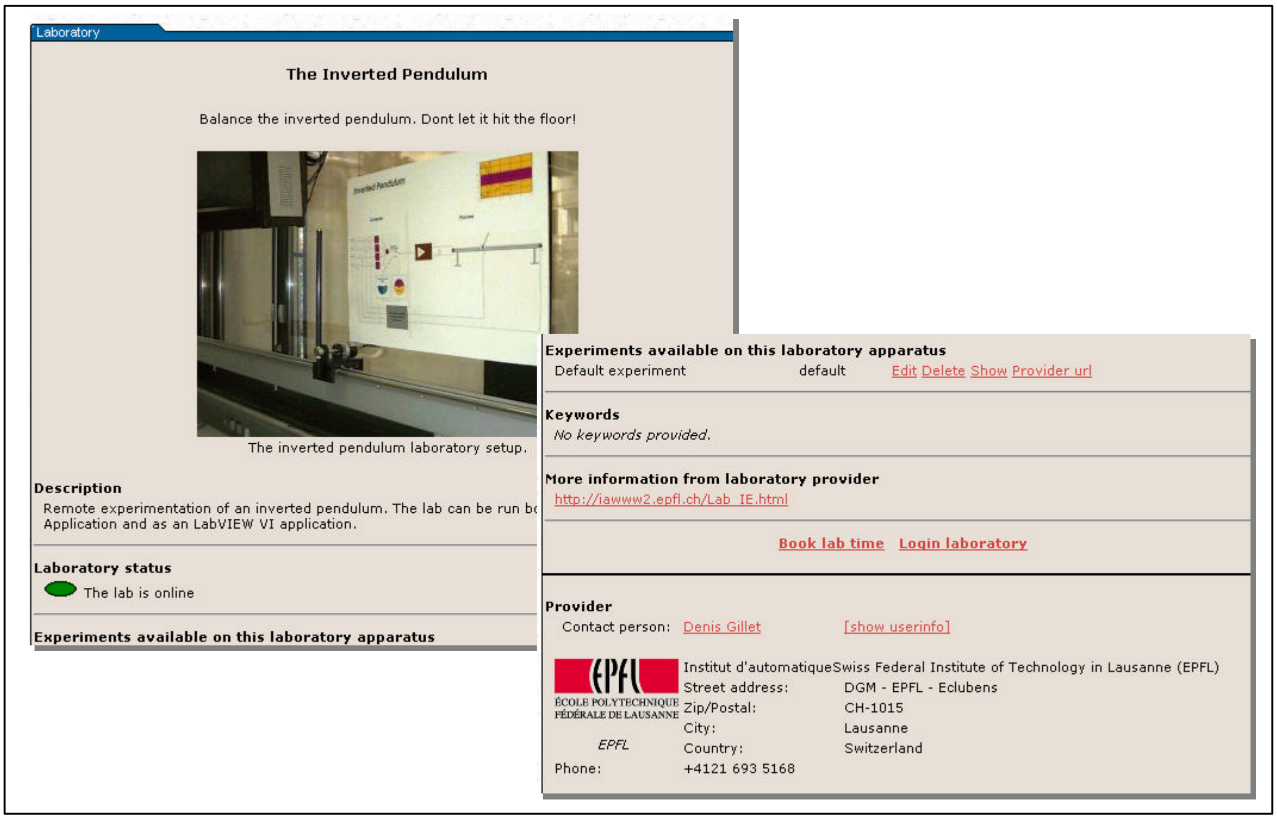
Some main characteristics of the Portal are:

- The Portal can display live status data retrieved from the Lab Server via the Cyberlab Communication Server.
- Information regarding the individual labs is provided via some unified Lab presentation pages.
- The Provider Admins can edit information regarding their labs via dedicated templates.
- The Provider Admins do some online configuration regarding access modes and access restrictions for their own labs.


Address http://www.cyberlab.org/frames.php
Go
CYBERLAB
About Cyberlab.Org
Labs
Booking
User
Misc.
search
participant
Differentiated access levels
User Registration
Login
Lab Browsing
Current labs
Actual status of lab (offline, available, occupied, etc

Provider.	Description.	Status.	
NTNU - Cybernetics	**Head box** Head box laboratory experiment	🟥	Show
	Refrigeration process Industry style refrigeration cycle process lab with control engineering experiments	🟩	Show
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE EPFL	**The Inverted Pendulum** Balance the inverted pendulum. Dont let it hit the floor!	🟩	Show Edit Delete
	The Electrical Drive	🟥	Show Edit Delete
	Optical Tracking System with PID control Optical tracking laboratory experiment	🟩	Show

Figure 2 Cyberlab Portal with differentiated access levels.



Laboratory

The Inverted Pendulum

Balance the inverted pendulum. Dont let it hit the floor!

The inverted pendulum laboratory setup.

Description
Remote experimentation of an inverted pendulum. The lab can be run b
Application and as an LabVIEW VI application.

Laboratory status
● The lab is online

Experiments available on this laboratory apparatus

Experiments available on this laboratory apparatus

Default experiment	default	Edit Delete Show Provider url

Keywords
No keywords provided.

More information from laboratory provider
http://iawww2.epfl.ch/Lab_IE.html

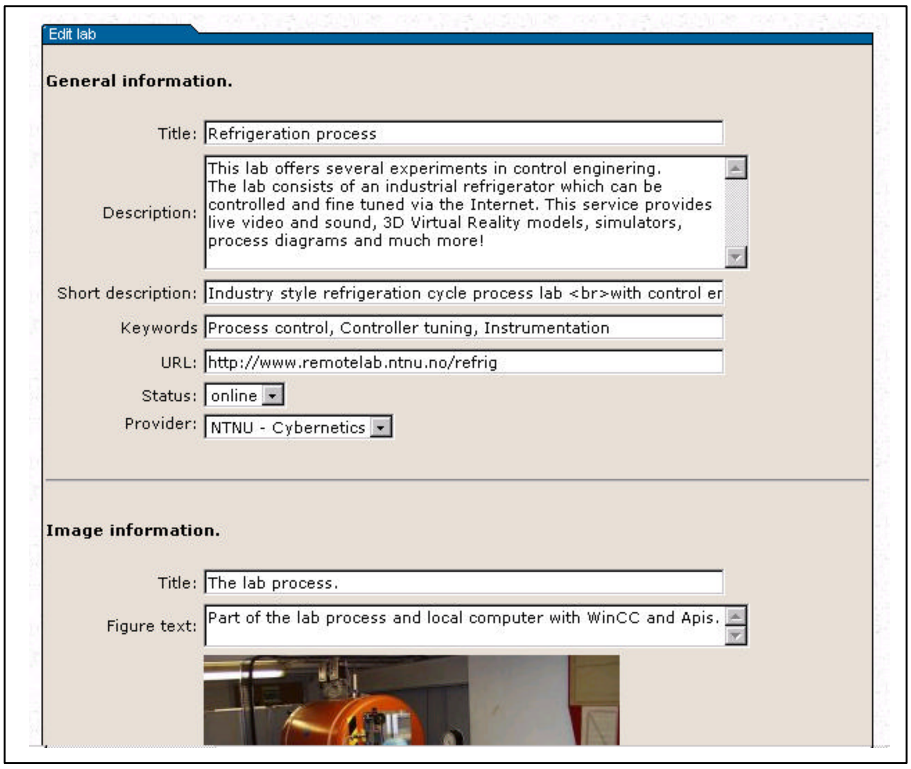
Book lab time Login laboratory

Provider
Contact person: Denis Gillet [show userinfo]

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EPFL

Institut d'automatique	Swiss Federal Institute of Technology in Lausanne (EPFL)
Street address:	DGM - EPFL - Eclubens
Zip/Postal:	CH-1015
City:	Lausanne
Country:	Switzerland
Phone:	+4121 693 5168

Figure 3 A unified Lab presentation



Edit lab

General information.

Title:	Refrigeration process
Description:	This lab offers several experiments in control enginering. The lab consists of an industrial refrigerator which can be controlled and fine tuned via the Internet. This service provides live video and sound, 3D Virtual Reality models, simulators, process diagrams and much more!
Short description:	Industry style refrigeration cycle process lab with control er
Keywords	Process control, Controller tuning, Instrumentation
URL:	http://www.remotelab.ntnu.no/refrig
Status:	online
Provider:	NTNU - Cybernetics

Image information.

Title:	The lab process.
Figure text:	Part of the lab process and local computer with WinCC and Apis.

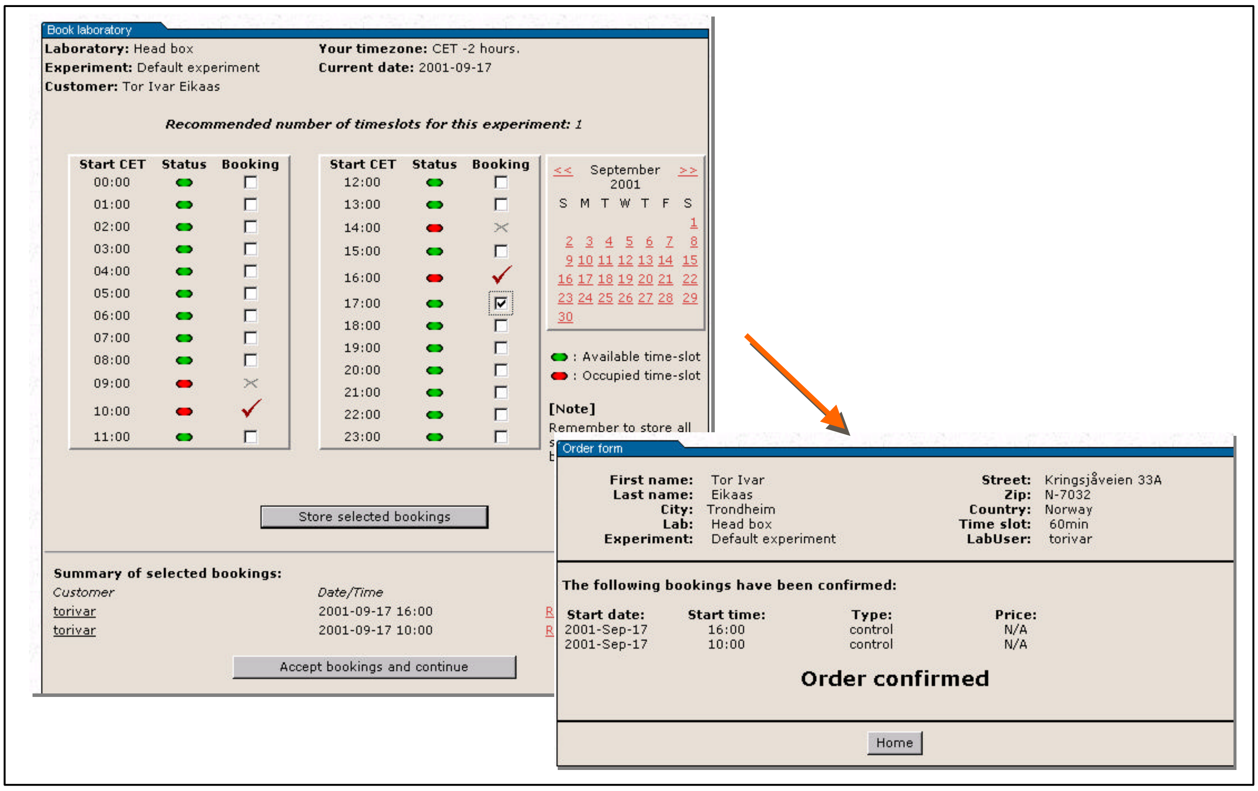
Figure 4 Provider Admin's online editing of Lab details

2.3 Booking options

The main booking options that is available via the Portal are:

- Timeslot based booking
- Priority based
- Recurring booking
- Bulk booking

In addition, the Provider Admins can easily configure the lab availability for external usage and own administrative bookings for their own labs.



Laboratory: Head box
Experiment: Default experiment
Customer: Tor Ivar Eikaas

Your timezone: CET -2 hours.
Current date: 2001-09-17

Recommended number of timeslots for this experiment: *1*

Start CET	Status	Booking
00:00	🟩	☐
01:00	🟩	☐
02:00	🟩	☐
03:00	🟩	☐
04:00	🟩	☐
05:00	🟩	☐
06:00	🟩	☐
07:00	🟩	☐
08:00	🟩	☐
09:00	🟥	✕
10:00	🟥	✓
11:00	🟩	☐

Start CET	Status	Booking
12:00	🟩	☐
13:00	🟩	☐
14:00	🟥	✕
15:00	🟩	☐
16:00	🟥	✓
17:00	🟩	☑
18:00	🟩	☐
19:00	🟩	☐
20:00	🟩	☐
21:00	🟩	☐
22:00	🟩	☐
23:00	🟩	☐

<< September 2001 >>

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

🟩 : Available time-slot
🟥 : Occupied time-slot

[Note]
Remember to store all
s
b

Summary of selected bookings:

Customer	*Date/Time*	
torivar	2001-09-17 16:00	R
torivar	2001-09-17 10:00	R

First name:	Tor Ivar	**Street:**	Kringsjåveien 33A
Last name:	Eikaas	**Zip:**	N-7032
City:	Trondheim	**Country:**	Norway
Lab:	Head box	**Time slot:**	60min
Experiment:	Default experiment	**LabUser:**	torivar

The following bookings have been confirmed:

Start date:	Start time:	Type:	Price:
2001-Sep-17	16:00	control	N/A
2001-Sep-17	10:00	control	N/A

Order confirmed

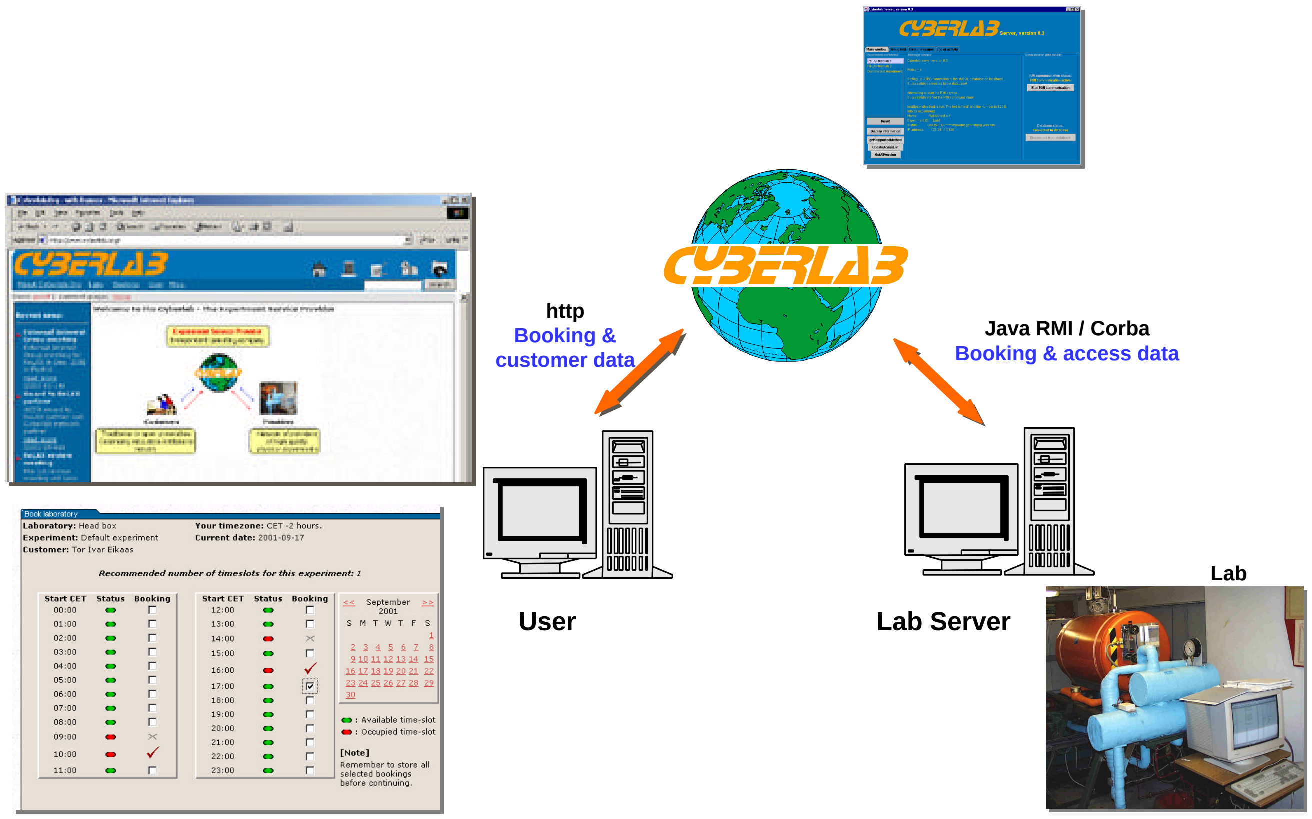
Figure 5 Booking options

2.4 The Booking process

The figure below illustrates the main information flow during the "Booking process", i.e. when the user is making a reservation for running a lab available in the Cyberlab Network.

Some main characteristics of the information flow at this stage are:

- At this stage, all information regarding a lab is retrieved from the Cyberlab DB.
- All requests from the User are done via the Cyberlab Portal. No communication is done directly between the User and the Lab Server.
- In order to make reservations for a lab run, the User must be registered as a Cyberlab User and given the appropriate privileges.
- The bookings are stored in the Cyberlab DB.
- Several booking modes exist as briefly described elsewhere in this note.
- Several alternative solutions and options are available with respect to how the requested bookings are sent to the Lab Server.



Laboratory: Head box **Your timezone:** CET -2 hours.
Experiment: Default experiment **Current date:** 2001-09-17
Customer: Tor Ivar Eikaas

Recommended number of timeslots for this experiment: 1

Start CET	Status	Booking	Start CET	Status	Booking
00:00	🟩	☐	12:00	🟩	☐
01:00	🟩	☐	13:00	🟩	☐
02:00	🟩	☐	14:00	🟥	✗
03:00	🟩	☐	15:00	🟩	☐
04:00	🟩	☐	16:00	🟥	✓
05:00	🟩	☐	17:00	🟩	☑
06:00	🟩	☐	18:00	🟩	☐
07:00	🟩	☐	19:00	🟩	☐
08:00	🟩	☐	20:00	🟩	☐
09:00	🟥	✗	21:00	🟩	☐
10:00	🟥	✓	22:00	🟩	☐
11:00	🟩	☐	23:00	🟩	☐

<< September 2001 >>

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

🟩 : Available time-slot
🟥 : Occupied time-slot

[Note]
Remember to store all selected bookings before continuing.

Figure 6 The Booking process

2.5 Running an experiment

When a User is running an Experiment on a Lab, there are two main information channels and one optional. These are illustrated in Figure 3 below.

Some main characteristics at this stage are:

- Normally, the User will log into the Lab directly via the Provider's web-site. The Provider will use Booking data received from Cyberlab during the Booking process in order to decide if the User should be granted access or not. Some options and alternative solutions do however exist.
- The User will interact with the Lab using a Provider specific solution for the GUI of the lab. This can be a downloaded application, a pure web-based solution or a mix. This lab GUI solution is normally developed and maintained by the Provider.
- The main part of the communication will take place directly between the User and the Lab Server, ref. the "meas, control, audio, video" label in the figure below.
- During an Experiment run, some administrative data can be retrieved by Cyberlab from the Lab Server, ref. the "Status & control" label. Several options and alternatives can be configured for this logging of data.
- During a Lab session, an optional status window (Cyberlab Status Window) can be used and configured by the Provider. This applet will provide the User with some additional lab and booking information sent directly to the User from Cyberlab as illustrated with the "Lab status" label.

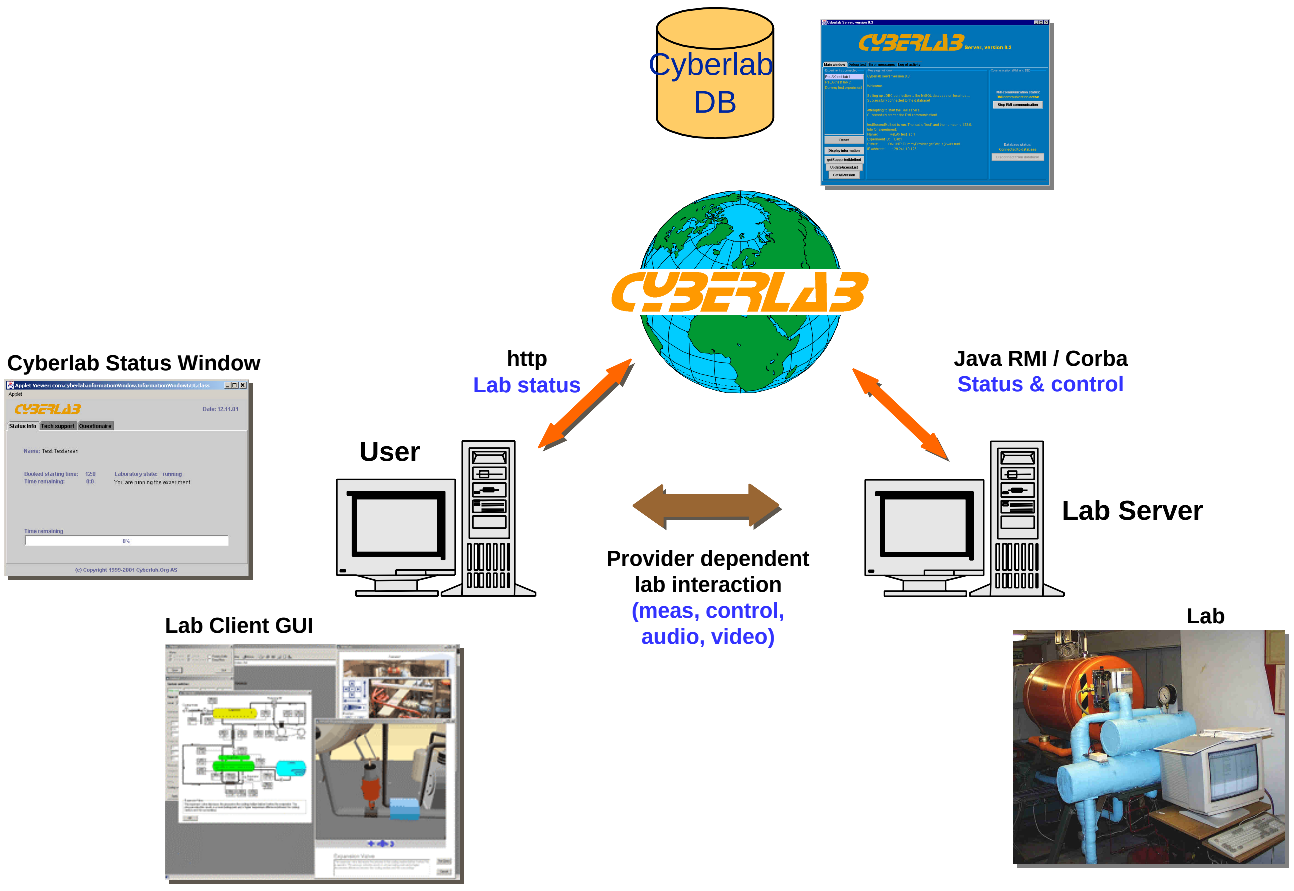


Figure 7 Running an Experiment on a Lab

2.6 Technical schematic

A more detailed schematic of the overall architecture of the Cyberlab System is given in Figure 8 below. As it can bee seen from this figure, the communication between Cyberlab and the Provider is implemented using Java RMI. Another alternative that is available today is a CORBA based solution.

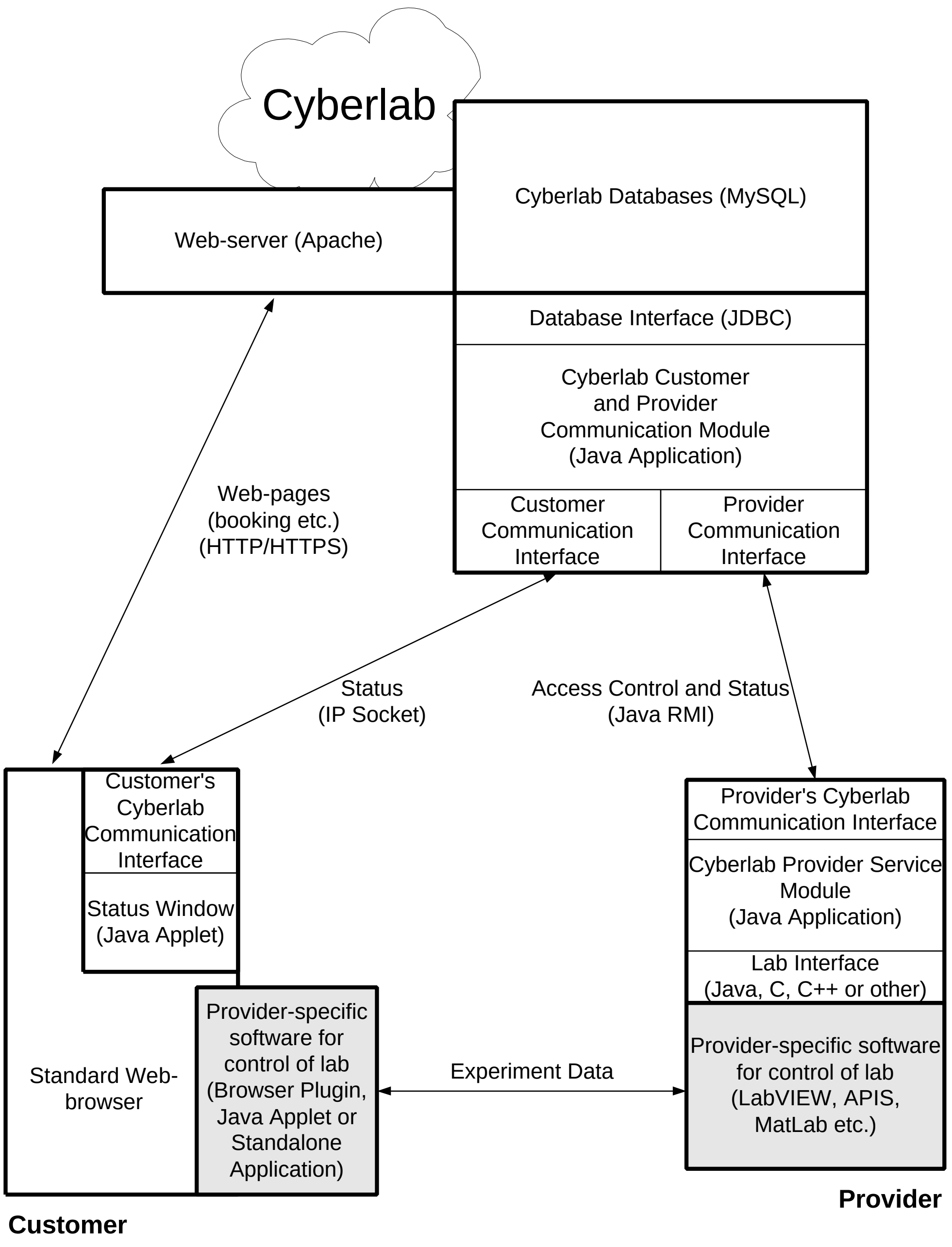
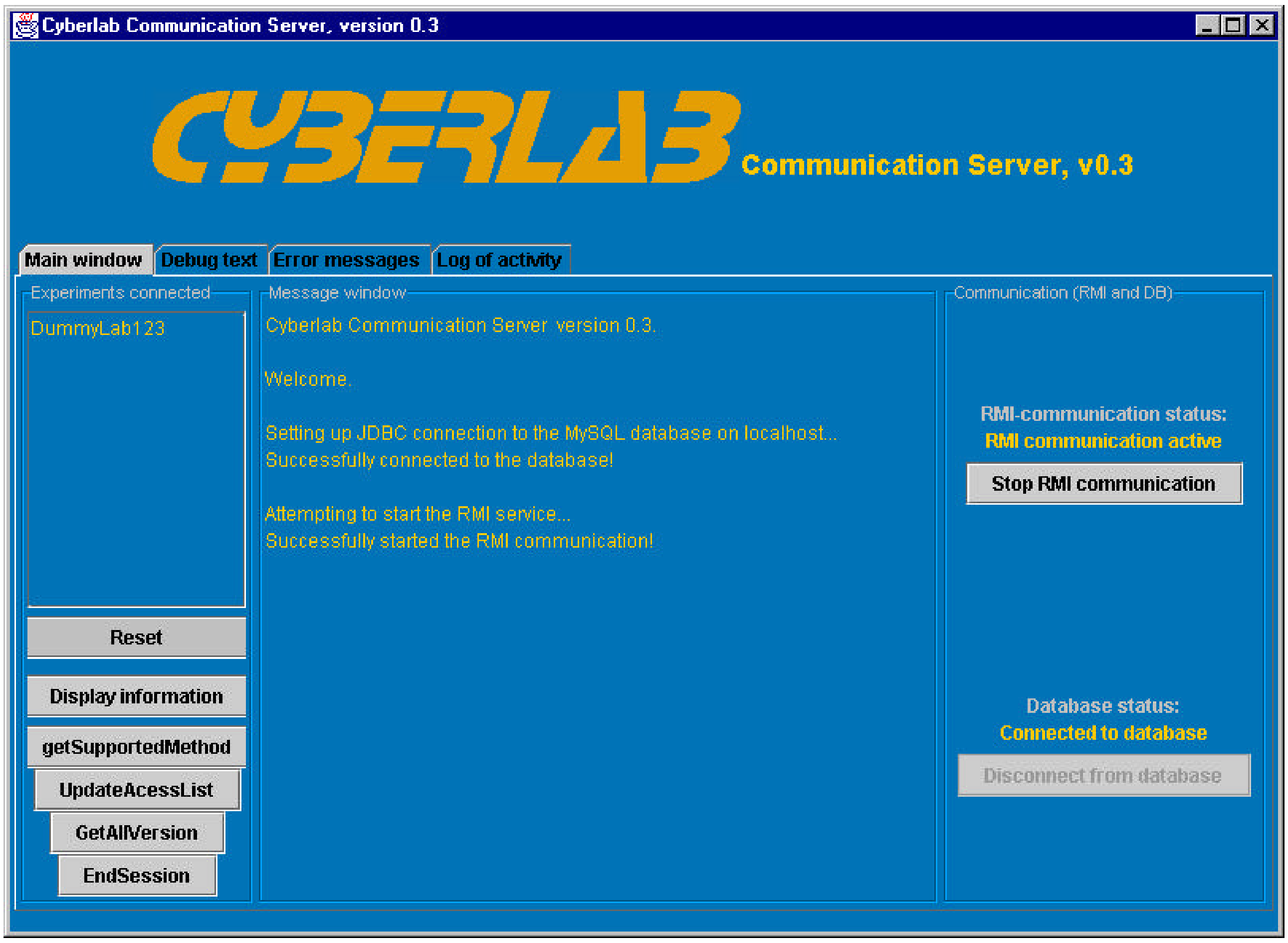

Cyberlab
Web-server (Apache)
Cyberlab Databases (MySQL)
Database Interface (JDBC)
Cyberlab Customer and Provider Communication Module (Java Application)
Customer Communication Interface
Provider Communication Interface
Web-pages (booking etc.) (HTTP/HTTPS)
Status (IP Socket)
Access Control and Status (Java RMI)
Customer's Cyberlab Communication Interface
Status Window (Java Applet)
Standard Web-browser
Provider-specific software for control of lab (Browser Plugin, Java Applet or Standalone Application)
Experiment Data
Provider's Cyberlab Communication Interface
Cyberlab Provider Service Module (Java Application)
Lab Interface (Java, C, C++ or other)
Provider-specific software for control of lab (LabVIEW, APIS, MatLab etc.)
Customer
Provider

Figure 8 A simple technical schematic of the main SW components and interactions for a typical solution using Java RMI.

3 The Cyberlab Communication Server (CCS)

The Cyberlab Communication Server (CCS) is implemented as a stand-alone Java application. The CCS runs on the same PC as the Cyberlab Web Server and the DB Server. The CCS communicates with the Cyberlab DB via a JDBC interface.

Today, the CCS is running on a Linux-based PC with Apache web-server and mySQL DB.



```
Cyberlab Communication Server  version 0.3.

Welcome.

Setting up JDBC connection to the MySQL database on localhost...
Successfully connected to the database!

Attempting to start the RMI service...
Successfully started the RMI communication!
```

Figure 9 Main window for Cyberlab Communication Server

The Cyberlab Communication Server is running 24/7 and servers as the connection point for Lab Servers that connects to the Cyberlab Network.

4 The Middleware solution

4.1 Using a database as a component in the middleware solution

If the Lab Server does not have a flexible Java interface, this is probably the easiest and (currently) the most stable way to integrate a new Lab with the Cyberlab Network.

The solution is illustrated in the figure below. Using this approach, the Provider only has to implement some DB access methods and implement the business logic for responding to messages sent from Cyberlab and initiating requests to Cyberlab.

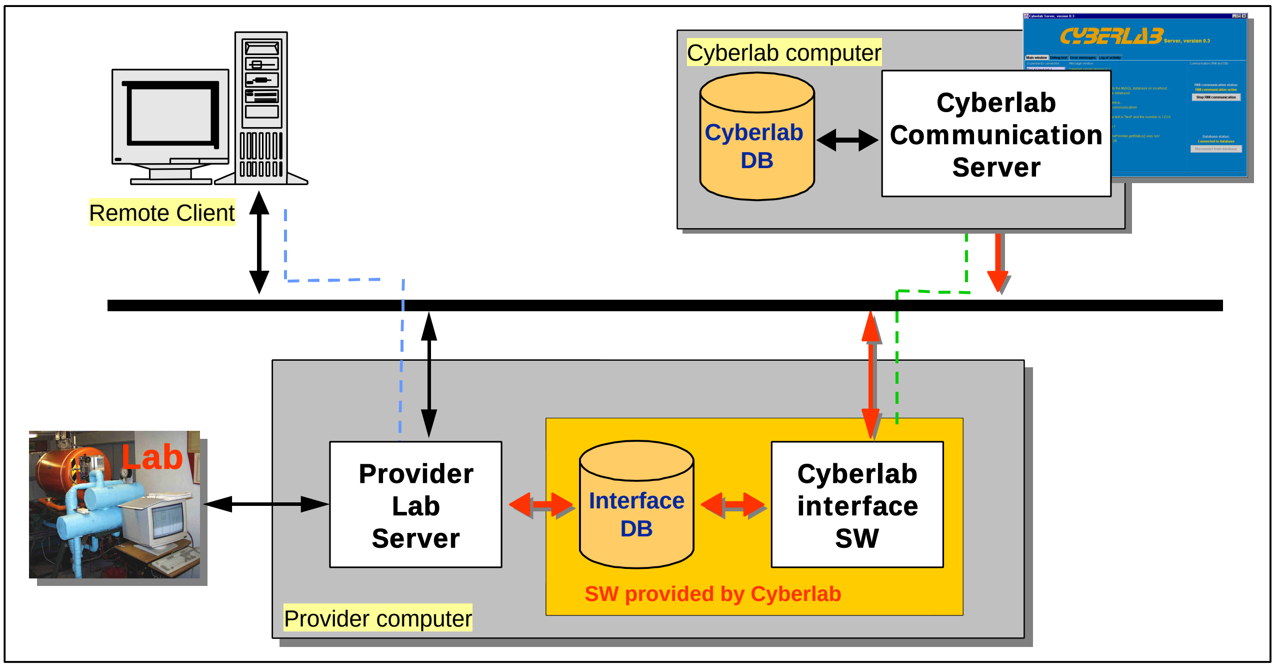


Figure 10 Using a DB in the middleware solution

4.2 The standard Java RMI version

The middleware solution based on Java RMI communication has been available for some time now, and has proved both stable and robust. This is the preferred solution if the Provider Lab Server is written in Java or if a flexible Java interface from the Lab Server already exists.

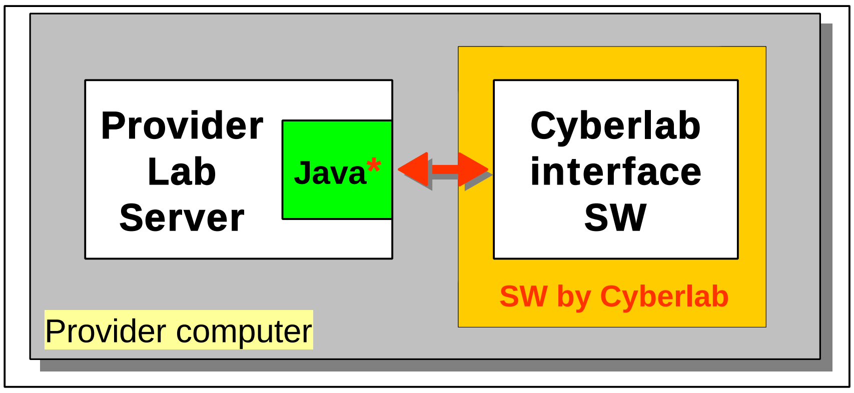


Figure 11 Using the standard Java solution

The main SW components using this solution is illustrated in the figure below, assuming a Java based Lab Server.

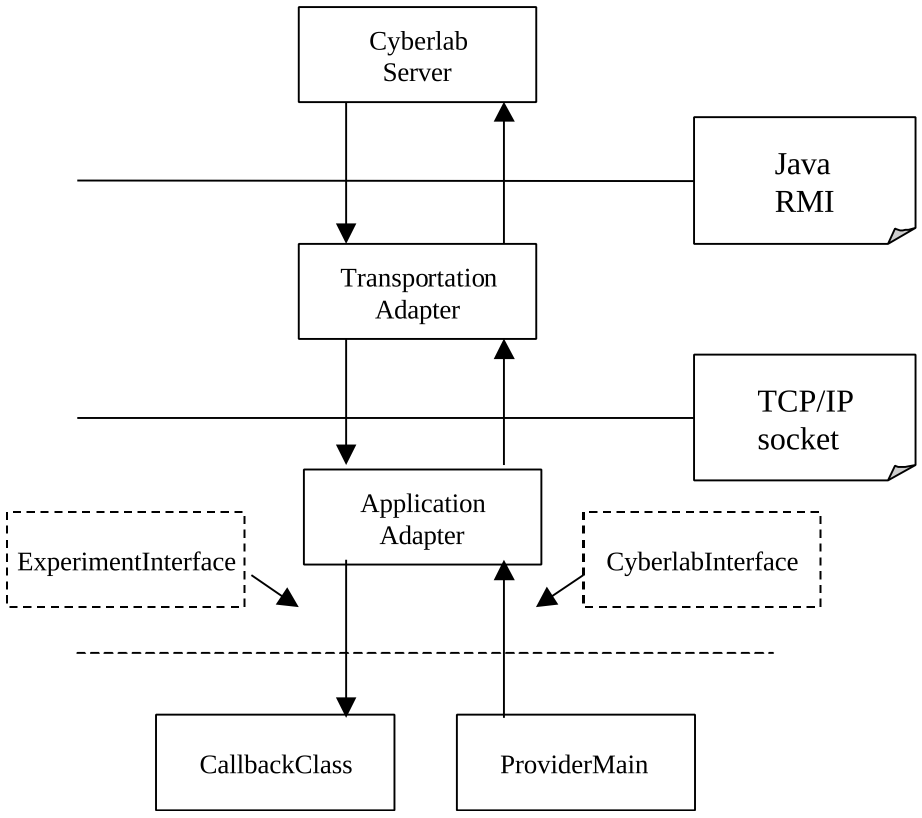


Figure 12 A simplified diagram of the standard middleware solution and its components

The two (Java) interfaces that the Provider must implement in his code are marked in the figure above as *ExperimentInterface* and *CyberlabInterface*.

In the following sections, a simplified description is given on how the information flows when method calls are made from the Lab Server (ProviderMain) to the Cyberlab Communication Server (called CyberlabServer in the figure above) and vice versa.

4.2.1 Calls initiated by the Lab Server

(Calls from ProviderMain to CyberlabServer)

A simplified description of the calls from ProviderMain to the CyberlabServer is given below.

ProviderMain

1. ProviderMain sends a call to ApplicationAdapter.
2. ApplicationAdapter sends the call (over TCP/IP SocketCommunication) to the TransportationAdapter.

TransportationAdapter

3. TransportationAdapter sends the call over Java RMI to the CyberlabServer.

CyberlabServer

4. CyberlabServer performs the necessary action based on the call (updating the DB etc).
5. CyberlabServer returns a return value over Java RMI to the TransportationAdapter.

TransportationAdapter

6. TransportatioAdapter returns the return value over the TCP/IP Socket communication further to the ProviderMain’s ApplicationAdapter.

ProviderMain

7. ApplicationAdapter passes the return value further to the original caller: ProviderMain.

4.2.2 Calls initiated by the Cyberlab Communication Server

(Calls from CyberlabServer to ProviderMain)

A simplified description of the calls from the CyberlabServer to the ProviderMain is given below.

CyberlabServer

1. CyberlabServer sends a call to the TransportationAdapter over Java RMI.

TransportationAdapter

2. TransportationAdapter sends the call further over TCP/IP Socket communication to the ProviderMain’s CallbackClass.

ProviderMain

3. CallbackClass does the necessary action(s) according to the call.
4. CallbackClass returns a return value over TCP/IP Socket communication to the TransportationAdapter.

TransportationAdapter

5. TransportationAdapter passes the return value further over Java RMI to the CyberlabServer.

CyberlabServer

6. CyberlabServer receives the return value.

4.3 The extended version

In parallel, Cyberlab has also developed an extension to the standard version. This new solution opens up for easy integration of CORBA and RMI/IIP solutions. This will give greater flexibility for alternative solutions of the Lab Server SW. In addition, a very nice feature with this design is that all the existing Providers using the old (but definitely not outdated) Java RMI solution will continue working without any modifications whatsoever.

The new addition to the middleware solution is currently under testing.

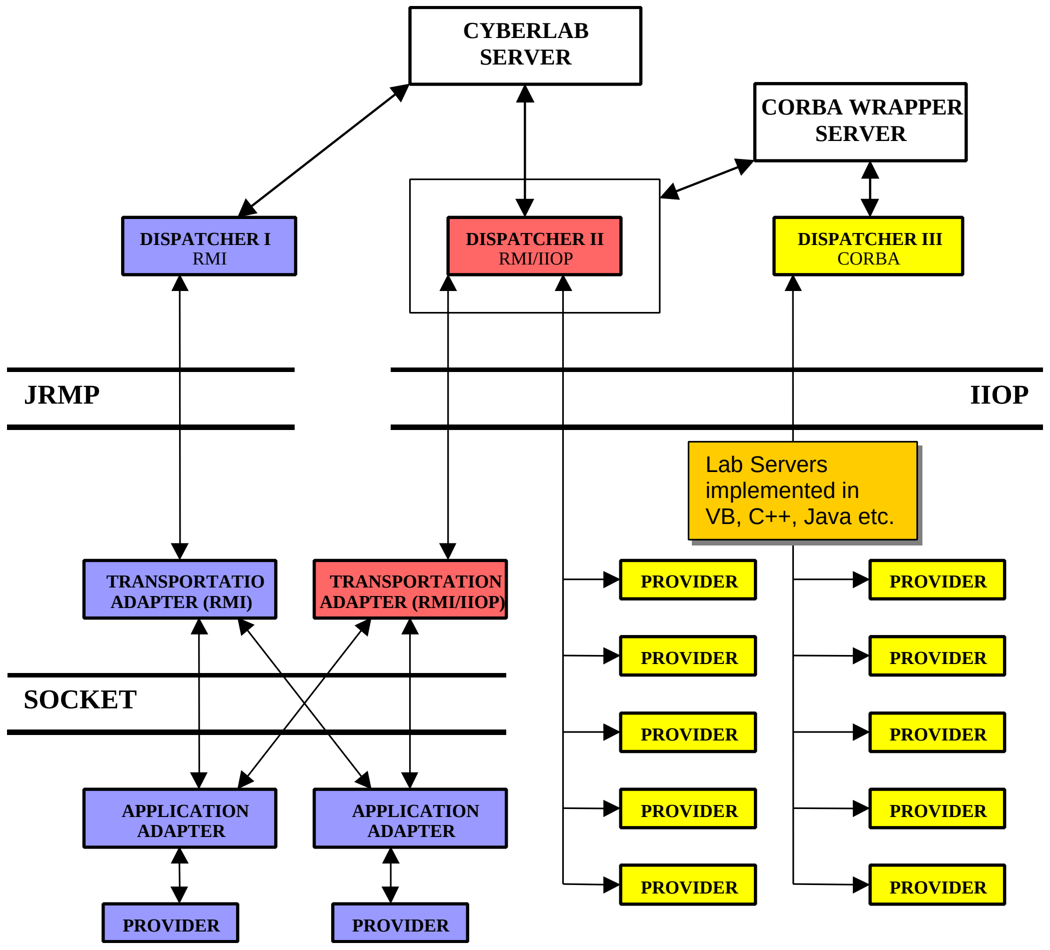


Figure 13 The extended middleware solution

The double lines marked JRMP (Java Remote Method Protocol) and IIOP (Internet Inter-ORB Protocol) is the separation between server and provider software. Everything below these lines is part of the provider software and hence situated on the provider computers.

Below are some notes regarding the various components in the extended middleware solution

- Cyberlab Server
  The server can handle communication over different protocols such as JRMP and IIOP.
- Dispatcher I – RMI
  This represents a dispatcher object registered with the Cyberlab Server. The object enables RMI (Remote Method Invocation) communication between software written in the Java programming language. This dispatcher is part of the standard solution, and is still is part of the

extended solution because of backward compatibility. It enables existing Providers to communicate with the enhanced server without the need of any new software.

- Dispatcher II – RMI/IIOP
  This is one of the dispatcher object in the extended version, and it can be contacted over the IIOP protocol. The object can be seen on as a RMI/IIOP server that can communicate with both RMI/IIOP and IDL (Internet Definition Language) clients. RMI/IIOP clients enable the same Java package as the server, which means they must also be written in Java. The IDL clients are language independent and can be written in any programming language that implements the Common Object Request Broker Architecture (CORBA). However, there still exist some compatibility problems between some of the existing CORBA implementations and the RMI/IIOP technology.
- CORBA Wrapper Server and Dispatcher III – CORBA
  These two components are used to avoid the compatibility problems that can occur between some CORBA implementations and the RMI/IIOP dispatcher object. The *CORBA Wrapper Server* act as a RMI/IIOP client towards the RMI/IIOP dispatcher object and at the same time it acts as a CORBA server implemented in Java. There are no compatibility problems between a RMI/IIOP and IDL communication when both client and server are written in the Java API. The communication between the dispatcher object in CORBA (Java IDL) and other CORBA implementations are handled without problems.
- Transportation Adapter (RMI) and Application Adapter
  These blocks are intermediate communication adapters used to enable provider software implemented in other languages than Java to communicate with the Cyberlab Server. These adapters are part of the standard solution with RMI communication model to free the solution from the Java language dependency. A provider using these adapters can still communicate with the extended version as long as the RMI dispatcher is registered with the server. The *Transportation Adapter* is written in Java and act as a client towards the RMI dispatcher on the server side, and at the same time it registers a Socket server within itself to talk with *Application Adapters* written in any language that support socket communication. The drawback is that application adapters will have to be implemented in the different languages and this is not a straightforward operation. Socket communication is low level programming compared with program packages such as RMI, RMI/IIOP and CORBA. Today Cyberlab.Org has implemented Application Adapters in Java and the C++ programming languages.
- Transportation Adapter (RMI/IIOP)
  This block represents a modified Transportation Adapter that will enable Application Adapters/Providers using the standard solution to connect and communicate through the RMI/IIOP dispatcher object. By using this object, the RMI dispatcher object at the server will be surplus to requirements, but it also means that already existing providers must change to this new adapter before communication can take place. In a transition state both communication dispatcher are registered on the server. This is also used in order to gain some experience with the extended solution.
- Provider
  These are the actual components of the Lab Servers that must be implemented by the lab providers. The two provider boxes to the left must have an Application Adapter written in the same programming language, while the rest are CORBA providers that need a CORBA implementation for the programming language in use. Today there exist CORBA implementations for all the major programming languages. Cyberlab has built test providers using the programming languages Java, C++ and Visual Basic. The solution seems to function properly.

The main advantages with the extended communication model are: Better language independence when integrating new Providers. The code on the client/provider side should also be somewhat easier to implement. It is also important that it works as an enhancement and not a logic change of the system, meaning that the old functionality is preserved while new functionality is added.

5 Connecting a Lab to the Cyberlab Network

5.1 Introduction

This chapter describes how Providers can use software provided by Cyberlab to connect a Lab to the Cyberlab System. This document focuses on the way it is done using Java. Other interfaces like C++ and Visual Basic are also available. It is assumed that the reader is comfortable with the Java programming language.

5.2 General requirements

- The existing (or to be created) Lab Server must be able to use Java. This does not necessarily mean that the whole Lab Server must be implemented in Java, but it must be able to communicate with a Java Application that can be an integrated part of the Lab Server.
- Other interfaces , like C++ and VB are available on request. Please contact Cyberlab for further details.
- Cyberlab must be notified about general information about the experiment (name, IP address, description etc.) for use with the Cyberlab database and for use on authenticating an Lab Server's connection to the Cyberlab Communication Server.

5.3 SW provided by Cyberlab

The files needed for a Provider are:

- ApplicationAdapter.jar
- TransportationAdapter.jar
- Crypt.jar
- DummyProvider.jar
- Source code for the interfaces
- CLProvider.ini
- CLProviderCallback.ini
- (startTransportationAdapter.bat)
- manifestForDummyProvider
- policyForExperiment
- Relevant JavaDoc

Provider example code:

- Source code for dummyProvider
  DummyProvider.jar
- startDummyProvider.bat
- Relevant JavaDoc for dummyProvider

All these files are available for the Providers from Cyberlab.

5.4 Other SW needed

It is required that the following software is installed:

- Java Development Kit (preferably version 1.4, but earlier versions are known to work properly)
- All software which is a part of the Provider's Lab Server

5.5 Connecting to Cyberlab Communication Server

The Cyberlab Communication Server currently running at all times. Please make an appointment with a Cyberlab Administrator in order to run some tests and verify the communication between your Lab Server and Cyberlab.

Please also refer to the following sections describing methods and configuration files needed when connection the Lab to www.cyberlab.org.

6 The Cyberlab – Provider Interface Methods

6.1 Introduction

This chapter describes how Providers can use software provided by Cyberlab to connect a new Lab and Lab Server to the CCS at the Cyberlab System.

6.2 General requirements

- The Lab Server must be able to use Java or one of the other languages supported by the various interfaces. Currently, Java and C++ is supported for the standard middleware solition, and Java, C++ and VB for the extended middleware solution. This does not necessarily mean that the whole Lab Server must be implemented in for example Java, but it must be able to communicate with a Java Application that can be assembled as an integrated part of the Lab Server.
- The integration between the Lab Server and the Cyberlab System can be made even less tight using a SQL DB as a middle layer between the Lab Server and a communication module provided by Cyberlab.
- Cyberlab must be notified about general information about the lab (name, IP address, description etc.) for use with the Cyberlab database and for use when authenticating a Lab Server's connection to the Cyberlab Server.

6.3 Interface methods

The following two sections describes the methods defined for the Cyberlab Communication Interface between the Cyberlab Communication Server and the Lab Servers.

The methods are categorized according the following criteria:

• *compulsory*	-	All experiment servers have to implement these methods
• *optional*	-	These may or may not be implemented by the Provider in the experiment server.
• *not used*	-	May have been used in a previous version. Not expected that it is needed anymore. Still kept in the interface in order to avoid unnecessary changes by the Provider.

6.3.1 Methods initiated by the Cyberlab Server

interface ExperimentInterface

This Interface describes all possible calls from the Cyberlab Communication Server (CCS) to the Lab Server. The methods are initiated by the CCS and sent to the Lab Server.

Table 1 ExperimentInterface - Methods initiated by Cyberlab

Method initiated by Cyberlab	Description
Compulsory	
getStatus()	Gets the actual status information from the laboratory. The status String **MUST** start with either: - "RUNNING: " - "ONLINE: " - "OFFLINE: " - "UNKNOWN: " The status should be followed by a textual description (e.g. "OFFLINE: Under

Method initiated by Cyberlab	**Description**
	maintenance"). **Returns:** Returns the status string followed by a description.
getSupportedMethods()	Method that returns the methods supported by the Lab Server. **Returns:** Returns the methods represented as a String, or "ERROR: " followed by an error description.
getVersion(identifier)	Returns a description of the current SW version for the specified SW components at the Lab Server. Arguments(identifier): TA – Transportation Adapter AA – Application Adapter ES - Experiment Server All – All available SW components **Parameters:** `identifier` - The identifier for the specified SW components **Returns:** the SW version(s) represented as a String, or "ERROR: " followed by an error description.
updateAccessControlList(accessList)	Method that updates the access control list (the bookings) from the Cyberlab DB. Cyberlab Server sends a list of accepted upcoming sessions for the specific lab. **Format:** Alternative 1 (default configuration): *username, (password), starttime, endtime, (priority);* ... For some labs, the format is: Alternative 2: *username, (password), starttime, endtime, (priority), (expID);* ... In the next release, all labs will use the 2nd format. **Parameters:** `accessList` - List of accepted upcoming sessions **Returns:** Returns"OK:", or "ERROR: " followed by an error description
endSession()	Method that ends any active user session. **Returns:** "OK", or "ERROR: " followed by an error description
getCurrentUTC()	Method that returns current UTC as seen by the Provider Lab Server. **Returns:** the value, represented as a String, or "ERROR: "

Method initiated by Cyberlab	**Description**
	followed by an error description
getExperimentId()	Gets the experiment ID (the Lab ID) from the Lab Server. This ID is used internally by the Cyberlab communication sw. **Returns:** the experiment ID from the Lab, represented as a String, or "ERROR: " followed by an error description.

Optional	
getExperimentValue(identifier)	Gets a boolean, numerical or textual value (as a String) from the Lab. The identifier is Provider-specific. **Parameters:** `identifier` - The identifier of the value to read (e.g. variable name) **Returns:** the value, represented as a String, or "ERROR: " followed by an error description
setExperimentValue(identifier, value)	Sets a numerical or boolean value for the Lab. The identifier is Provider-specific. **Parameters:** `identifier` - The identifier of the value to read (e.g. variable name) `value` - The new value to set to the variable. Boolean is represented as 0.0=false and all other numbers = true. **Returns:** the value that was set if successful, or "ERROR: " followed by an error description
resetExperiment()	Resets the Lab by setting it to a predefined and ready safe. **Returns:** "OK" if successful, or "ERROR: " followed by an error description. ***Note:*** It is strongly reccomended to implement this method.
connectionIsLost()	Method that gets called when the connection to the Cyberlab Communication Server is lost. **Parameters:** `reason` - The reason for the break in communication **Returns:** "OK", or "ERROR: " followed by an error description

Not used	
throwOutUser(sessionId)	Throws out a user from controlling or observing the experiment (*not used*).

Method initiated by Cyberlab	Description
	Parameters: `sessionId` - The session ID to identify the user to throw out **Returns:** "OK" if successful, or "ERROR: " followed by an error description
throwOutAllUsers()	Throws out all users from controlling or observing the Lab (*not used*). **Returns:** "OK" if successful, or "ERROR: " followed by an error description

6.3.2 Methods initiated by the Lab Server

interface CyberlabInterface

This Interface describes all possible calls from the Lab Server to the Cyberlab Communication Server (CCS). The methods are initiated by the Lab Server and sent to the CCS.

Table 2 CyberlabInterface - Methods initiated by Lab Server

Method initiated by Lab Server	Description
Compulsory	
connect()	Establish the connection with the CCS. If successful, the Laboratory status is set as ONLINE. The Provider does not need to do any implementation work for this method. **Returns:** "OK" to signal that the message was received, or "ERROR: " followed by an error description.
loginResult(labName,sessionID,result)	Method that returns the result from attempted login and the actual login time. **Parameters:** `LabName` – The Cyberlab identifier for this lab `sessionId` - A session ID describing the user `result` – result from attempted login **Returns:** A String describing the result from attempted login and actual login time, or "ERROR: " followed by an error description. This method is also used when users log out of the system.
isAlive()	Cyberlab System method that checks the Cyberlab connection. The Provider does not need to do any implementation work for this method. **Returns:** "OK" to signal that the message was received, or "ERROR: " followed by an error description.

Optional	

Method initiated by Lab Server	**Description**
getVer() Next release: getCyberlabVersion ()	Gets the current and most up-to-date version numbers of the main components in the Cyberlab system. This method is not a necessary function for the Cyberlab system, but is used mainly for maintenance and debugging. **Returns:** The version numbers of the components in the system, or "ERROR: " followed by error description. The main components: CCI Cyberlab Communication Interface (CyberlabInterface and ExperimentInterface) CCS Cyberlab Communication Server TA Transportation Adapter AA Application Adapter
disconnect(offlineStatusDescription)	Method that notifies the CCS that this experiment is about to disconnect and close the connection with the Cyberlab Server (*optional*). The Laboratory status will be set to OFFLINE. One must specify the reason for the OFFLINE status. **Parameters:** `offlineStatusDescription` - A String describing the OFFLFINE status. Typical values are: Maintenance, SW upgrade, etc. **Returns:** OK" to signal that the message was received, the last answer from Cyberlab, or "ERROR: " followed by an error description.
log(type, sessionId, action)	Method that logs some specific activity on the Lab Server. The activity can be initiated both by the user and by the system. **Parameters:** `type` - Log type. Legal values: USER, SYSTEM `sessionId` - A session ID describing the user **Returns:** A String describing the specific activity, or "ERROR: " followed by an error description.
checkUserAccess(sessionId)	Method to check if a specific user has access to the lab NOW. **Parameters:** `sessionId` – A session ID describing the user **Format:** <username>;<password> **Returns:** A String describing the access rights, or "ERROR: " followed by an error description.

Method initiated by Lab Server	**Description**
Not used	
getBookingData()	Method that gets the booking data. **Returns:** A String describing the booking data, or "ERROR: " followed by an error description.

7 Experiment Interface - Implementation details

The Experiment Interface is part of the Cyberlab Communication Interface. The Experiment Interface is defined via a Java Interface called ExperimentInterface. The Providers must provide an implementation of this Java interface.

These methods in the ExperimentInterface are initiated by the Cyberlab Communication Server at the Cyberlab site, and sent to the Lab Server (Experiment Server) at the Provider site.

A summary of the methods is given in the table below. Some implementation details and guidelines are also given in the following sections.

Table 3 ExperimentInterface – summary of methods

Method initiated by Cyberlab	
Compulsory	
getStatus()	
getSupportedMethods()	
getVersion(identifier)	
updateAccessControlList(accessList)	
endSession()	
getCurrentUTC()	
getExperimentId()	
Optional	
getExperimentValue(identifier)	
setExperimentValue(identifier, value)	
resetExperiment()	Strongly reccomended to implement this
connectionIsLost()	
Not used	
throwOutUser(sessionId)	
throwOutAllUsers()	

Note 1
The Cyberlab Communication Server must support both the Compulsory and the Optional methods.

Note 2
The enclosed example file CallbackClass.java, implementing the ExperimentInterface for a simple Provider Lab Server (dummyProvider), does not fully implement the specification of the return values described in the following sections.

Important files for the Provider

- The ExperimentInterface:
  filesForProvider\com\cyberlab\ ExperimentInterface.java
- Example of a Provider implementation of the ExperimentInterface:
  filesForProviders\code\com\cyberlab\dummyProvider\CallbackClass.java

7.1 Compulsory methods

7.1.1 getStatus

7.1.1.1 The method

public java.lang.String **getStatus**()

This method is compulsory, and is called by the Cyberlab Communication Server. The Lab Server returns the actual status information about the laboratory.

Parameters

```
no parameters
```

Return value

The Lab Server returns the status information in a String. The format of the status information String is:

<status>: <status description>

The legal values for <status> are defined in the table below. The purpose of <status description> is to provide some additional information underpinning the <status>. The content of <status description> is Provider and Lab dependent, but some suggestions and guidelines are given below.

<Note>	The <status> must be one of the legal ones defined in the table below.

Table 4 Legal values for the status field in getStatus

<status>	Description
RUNNING	This means that somebody has logged into the lab and is running the lab doing some experimentation on it. The <status description> should give more information like: “remote Cyberlab usage” “local usage”
ONLINE	The lab is available for remote access. A person with a valid username and password can log into the lab. The <status description> should give more information like: “Ready for remote access”
OFFLINE	The lab is not available for remote access. The <status description> should give more information like: “under maintenance. Expected to be online yyyy-mm-dd” “upgrading lab SW. Expected to be online yyyy-mm-dd”
UNKNOWN	It is not possible to determine the lab status. The <status description> should give more information like: “Lab is not responding” “No socket communication with the Lab Server” “Unknown reason for break in the communication”
UNAVAILABLE	The lab is currently not available for Cyberlab usage: “Lab is used locally”

7.1.1.2 CCS implementation details

- The method is run on all labs with a lab-specific frequency.
- The method can also be triggered manually or based on some specific event.
- The update interval is specified individually for each lab. The default is set to 10min.
- The system automatically checks the time deviation between the CCS and the Provider Lab Server. If the difference is too large, a notification is sent by email to the Provider admin.

7.1.1.3 Lab Server implementation details

All Lab servers must implement this method, but the business logic needed for defining the actual status will vary between the individual labs. For some Labs, it will be possible to check only if some program is running and probably the current measurement of a few tags. For other Labs, a more complicated logic must be specified and implemented in order to deduct the actual status.

7.1.1.4 Example code

N/A

7.1.2 getSupportedMethods

7.1.2.1 The method

public java.lang.String **getSupportedMethods**()

This method is compulsory, and is called by the Cyberlab Communication Server. The Lab Server returns a list with the methods supported by the Lab Server. The list should include the implementation status for all methods, both for the CyberlabInterface and in the ExperimentInterface and for both the compulsory and the optional methods.

Parameters

```
no parameters
```

Return value

The Lab Server returns the supported methods in a String or “ERROR: “ followed by an error description.

The format of the supported methods String is:

<interface_version>;<method_status_1>; … <method_status_N>;

where

<interface_version>	The Cyberlab Communication Interface version. This is represented like this: CCI version: <majorversion.minorversion> The current version is i.e. 0.5. The interface version is given by Cyberlab.
<method_status_i>	= <method>, <version>, <status >
<method>	The name of the method (getStatus, getSupportedMethods, etc)

<version>	Version number of implementation. This is a version number provided by the Provider. It is recommended to use the format <majorversion>.<minorversion>
<status>	This Implementation status of the individual method. This is a Provider-defined text indicating the actual status of the implementation. Typical values are:

"full"	Full implementation
"draft"	A test version
"dummy"	Only a sceleton
"no"	Not implemented.

7.1.2.2 CCS implementation details

- This method is normally called by the CCS on request only, and not with a fixed frequency, and is normally only used for debugging and statistical purposes.

7.1.2.3 Lab Server implementation details

- There should be no '\n' embedded in the text string.
- Below is an example on how the list of supported methods included in a String could be returned from the Lab Server. Only a part of the list is included:

```
CCI version: 0.3;getStatus,1.0,full;getSupportedMethod,0.9,draft;
getVersion,0.1,dummy; <stuff deleted> ;log,1.0,full;
```

7.1.2.4 Example code

N/A

7.1.3 getVersion

7.1.3.1 The method

```
public java.lang.String getVersion(java.lang.String identifier)
```

This method is compulsory, and is called by the Cyberlab Communication Server. The Lab Server returns a list with the version number of all or a specified component of the system.

Parameters

`identifier` The identifier for the specified SW components

TA	Transportation Adapter
AA	Application Adapter
[LS | ES]	Lab Server / Experiment Server
All	All available SW components

Return value

The Lab Server returns the version of all or the specified component in a String or "ERROR: " followed by an error description.

The format of the version String is:

Components:<component_1>; … <component_N>;

where

<component_i>	= <component_id>, <version>
<component_id>	The short name of the SW component (TA, AA, ES, All)
<version>	Version number of the SW component. This is a version number defined by Cyberlab for TA and AA and defined by the Provider for ES/LS. The version mumber follows the format <majorversion>.<minorversion>

7.1.3.2 CCS implementation details

♦ This method is normally called by the CCS on request only, and not with a fixed frequency, and is normally only used for debugging and statistical purposes.

7.1.3.3 Lab Server implementation details

♦ There should be no '\n' embedded in the text string.

♦ The version numbers are the ones defined by Cyberlab for TA and AA.

♦ Both ES and LS is accepted as a short notation for the Lab Server.

♦ The version number for LS / ES is defined by the Provider.

♦ Other return values may be specified by Cyberlab for Labs using other interfaces than the ones for Java.

♦ Below is an example on how the returned String could look like when returned from the Lab Server after requesting 'All':

```
TA,0.5;AA,0.5;ES,1.4;
```

7.1.3.4 Example code

N/A

7.1.4 updateAccessControlList

7.1.4.1 The method

public java.lang.String **updateAccessControlList**(java.lang.String accessList)

This method is compulsory, and is called by the Cyberlab Communication Server normally once every 12hr. The Cyberlab Communication Server sends a list of all the accepted upcoming sessions (bookings) for the next 24hrs for the specific lab.

Parameters

`accessList` List of accepted upcoming sessions

The format of the accessList String is:

<session_1>; … <session_N>;

where

Alternative 1:
<session_i> = *<username>, (<password>), <starttime>, <endtime>, (<priority>)*

Alternative 2:
<session_i> = *<username>, (<password>), <starttime>, <endtime>, (<priority>),(expID)*

<username>	The Cyberlab username identifying the user that has been booked for a lab session.
<password>	The password to be used during login on the Laboratory. Normally, the password is encrypted, but it can also be in clear-text for some labs. For some labs, the password is not sent via this mechanism. This parameter is optional (depending on the solution used at the individual lab).
<starttime>	The start time of the lab session. The time is given in UTC using the following format: YYYY-MM-DD hh:mm
<endtime>	The end time of the lab session. Same format as for the <starttime>
<priority>	The given priority for the actual session. This is normally represented as an integer in the range 1-10, with 1 as the highest priority. This parameter is optional, and normally only used for labs with a priority-based access control.
<expID>	The experiment identifier for current session. Interpreted as 'default experiment' if <expID> is not specified.

<NOTE> - The next release of the interface will only support alternative 2 above.
- Currently, each lab must be configured explicitly to support alternative 2. The default configuration is alternative 1.
- Contact Cyberlab in order to configure the lab to receive the booking data in the alternative 2 format.

<NOTE> - The time (starttime and endtime) is sent to the individual labs in UTC.

Return value

The Lab Server returns the update status as “OK:” if the access list has been successfully updated, or “ERROR: “ followed by an error description if the update has failed.

7.1.4.2 CCS implementation details

The following rules apply for the CCS implementation of the *updateAccessControlList* method:

- The booking data is collected from the Cyberlab booking database.
- The Cyberlab Communication Server normally calls this method every 12hr.
- All upcoming sessions for the specific lab for the next 24hrs are sent every time the method is invoked.
- The method is also invoked every time an existing booking for the specific lab has been modified (new, moved or deleted) from the Cyberlab Portal.

- By default, two or more consecutive sessions in the Cyberlab booking DB, is transferred to the Provider Lab Server as one continuous session. This in order to avoid the user to be thrown out after the end of the individual session(s). The criteria for concatenating consecutive bookings are a) same labuser b) the same lab and experiment ID.
- The labs can be individually be configured so that the CCS do not concatenate the bookings before sending them to the Lab Server.
- No spaces are allowed in the username.
- The usage of priority must be configured for each individual lab.
- The password can be sent to the Lab Server in clear text or using a built-in encryption algorithm.
- All start and stop times are stored and sent to the Provider in UTC.
- The bookings are moved to a historical DB after endtime has been reached.

7.1.4.3 Lab Server implementation details

- The actual implementation of this method is Lab specific.
- Some Labs are using a local SQL DB for storing the received bookings. mySQL has been used with success for this purpose by several Providers.
- The Cyberlab Crypt package can be used for the encryption / decryption of the passwords.
- The type of encryption to be used must be agreed with Cyberlab.

7.1.4.4 Example code

Here is a very simple example of a Provider implementation of the updateAccessCotrolList method.

```
   /**
    * Method is the implementation of the 'updateAccessControlList' of the
    * Experiment Interface. The method is the Lab Server respons when the method
    * is called from the Cyberlab Communication Server.
    * This is the LabXYZ specific implementation of how to handle the session-
    * file.
    * - The access control list is stored in a local 'DB' in a file called
    *   cl_session.
    *- The filename is specified by the 'Filename' entry in the configuration
    *   file CLProviderCallback.ini.
    * - The session-file is located in a protected area.
    *   The path (needed by the Lab Server) to the session file is specified by
    *   the registry-key:
    *      'My computer\HKEY_LOCAL_MACHINE\SOFTWARE\ServerXYZ\SessionPath'
    * - The Lab Server will use this information when using the OPC Security
    *   mechanism when clients are connecting to the lab via the (remote)
    *   application.
    * - The Lab Server is using the following format for the session file:
    *      username, password, starttime, endtime,
    *      ...<br>
    *      username, password, starttime, endtime,
    * - Currently, only the alternative 1 format of the access list is
    *   supported:
    *      session_i = username,(password),starttime,endtime,(priority)
    *   and not the new format:<br>
    *     session_i = username,(password),starttime, endtime,(priority),(expId)
    * - The time format (starttime and endtime) is:
    *     YYYY-MM-DD HH:mm:ss<br>
    * - The time is stored in UTC<br>
    *
    * @param accessList List of accepted upcoming sessions
    * @return  the value, represented as a String, or "ERROR: " followed by
    *       an error description
    */
   public String updateAccessControlList(String accessList) {
     // Creates BufferedWriter, used to write to file.
     BufferedWriter writer = null;
     try {
       writer = new BufferedWriter (new FileWriter (FileName));
     }
     catch (IOException e) {
       System.out.println("Can't open: " + FileName);
     }
     StringTokenizer token = new StringTokenizer(accessList, ";");
     while(token.hasMoreTokens()) {
       String type = token.nextToken();
       try {
         // Writes a line to file
         writer.write(type);
         writer.newLine();
       }
       catch (IOException e) {
         System.out.println("write failed for: " + FileName);
       }
     }

     // Write some additional entries for Admin purposes
     try {
       writer.write("<superuser>,XXX,2002-01-01 01:00:00,2002-12-31 23:59:00,");
       writer.newLine();
     }
     catch (IOException e) {
       System.out.println("write failed for: " + FileName);
     }

     // Close file
     try {
       writer.close() ;
     }
     catch (IOException e) {
       System.out.println("Can't close: " + FileName);
     }
   }
```

7.1.5 endSession

7.1.5.1 The method

public java.lang.String **endSession**()

This method is compulsory, and is called by the Cyberlab Communication Server. The Lab Server should respond to this method by ending any active user session on the Lab.

Parameters

```
no parameters
```

Return value

The Lab Server returns “OK” in a String or “ERROR: “ followed by an error description.

7.1.5.2 CCS implementation details

- ♦ This method is only called by the Cyberlab Communication Server on request or based on a given event.
- ♦ The method could be triggered as a response from a User that is not able to log into the Lab due to an existing session conflicting with a booked and accepted session.

7.1.5.3 Lab Server implementation details

Some implementation details for the business logic on the Provider Lab Server:

- ♦ The action of this method must be clarified for a couple of special cases:
  Should it be possible to end a session if
  - a) a local user is performing local operation of the Lab during a timeslot when a Cyberlab user has booked lab time?
  - b) a local user is performing local maintenance of the Lab?
  - c) the session is performed by a Provider Admin?
  - d) the session is outside a timeslot booked by a Cyberlab Customer?
  - e) the Lab is not defined as ONLINE or RUNNING?
- ♦ The actual business logic for this method should be agreed with Cyberlab.

7.1.5.4 Example code

N/A

7.1.6 getCurrentUTC

7.1.6.1 The method

public java.lang.String **getCurrentUTC**()

This method is compulsory, and is called by the Cyberlab Communication Server. The Lab Server should respond to this method by sending the current UTC as seen by the Lab Server.

Parameters

```
no parameters
```

Return value

The Lab Server returns current UTC in a String or “ERROR: “ followed by an error description.

The format of the current UTC String is:

<currentUTC>	YYYY-MM-DD hh:mm

7.1.6.2 CCS implementation details

- The CCS calls this method with a specified frequency.
- The period can be configured individually for each Lab.
- The default request frequency is 24hrs.
- The offset (error) is computed by the CCS. If the deviation (the error) is greater that a specified limit, an email is automatically sent to the Provider Admin.

7.1.6.3 Lab Server implementation details

- This method is important in order to detect and possibly correct problems with incorrect time settings at the Provider computer.
- If the time is not set correctly, the Lab Users will have problems accessing the Lab according the reservations that have been made.
- Some of the notations used:

Greenwich Mean Time	(GMT)
Coordinated Universal Time	(UTC)
Dayligth Savings Time	(DST)

Norway:	UTC+0200 (with DST)
	UTC+0100 (no DST)

- More information about the time zones can be found on

http://www.cl.cam.ac.uk/~mgk25/iso-time.html	Overview of timezones etc.
http://www.twinsun.com/tz/tz-link.htm	Timezone conversion

7.1.6.4 Example code

N/A

7.1.7 getExperimentId

7.1.7.1 The method

public java.lang.String **getExperimentId**()

This method is compulsory, and is called by the Cyberlab Communication Server. The Lab Server returns a String giving a unique identification of the Lab.

Parameters

```
no parameters
```

Return value

The Lab Server returns a unique identification of the Lab in a String or “ERROR: “ followed by an error description.

The unique identifier will follow the following syntax:

<providerShortName><labShortName>

Some typical identifiers are:

NTNUrefrig
RUBoptiCon
EPFLpendulum

7.1.7.2 CCS implementation details

N/A

7.1.7.3 Lab Server implementation details

- The Provider must only use the identifier provided by Cyberlab for the specific Laboratory as part of the agreement between Cyberlab and the Provider.
- The identifier is currently the defined in a string in the file *CLProviderCallback.ini* that must be located in the same directory as where the Provider Server is started from.

  Experiment_Id=<unique identifier>

- The variable name used for the identification will probably change in the next version of the Provider Interface SW.

A special note regarding the Java interface

- It should normally be sufficient for the Provider just to return the EXPERIMENT_ID in the implementation of getExperimentId() in CallbackClass.java.

7.1.7.4 Example code

N/A

7.2 Optional methods

7.2.1 getExperimentValue

7.2.1.1 The method

public java.lang.String **getExperimentValue**(java.lang.String identifier)

This method is optional, and is called by the CCS. The Lab Server returns the value for a specified identifier (variable name). The returned value can be a boolean, numerical or textual value (as a String) from the Lab.

Parameters

`identifier` The identifier of the value to read (e.g. variable name, tag)

Return value

The Lab Server returns the value, represented as a String, or "ERROR: " followed by an error description.

The format of the return String should follow the following syntax:

<identifier>:<value>

7.2.1.2 CCS implementation details

- This method can for example be used for generating customised reports for documenting an experimental run.

7.2.1.3 Lab Server implementation details

- The names of the identifiers are Provider specific.
- The Provider may restrict the access to a limited set of values.

7.2.1.4 Example code

N/A

7.2.2 setExperimentValue

7.2.2.1 The method

public java.lang.String **setExperimentValue**(java.lang.String identifier, double value)

This method is optional, and is called by the Cyberlab Communication Server. A new value for a specified identifier is specified from the Cyberlab Communication Server and sent to the Lab Server. The corresponding identifier at the lab is updated with the new value.

Parameters

`identifier`	The identifier of the value to read (e.g. variable name, tag)
`value`	The new value to set to the variable. Boolean is represented as 0.0=false and all other numbers = true

Return value

The Lab Server returns “OK: ” represented as a String, or "ERROR: " followed by an error description.

7.2.2.2 CCS implementation details

- A set of high level identifiers may be defined in the Cyberlab System.

7.2.2.3 Lab Server implementation details

- This method is important for Labs where customised reports are generated.
- The names of the identifiers are Provider specific.
- The Provider may restrict the access to a limited set of values.
- The rules for when and by whom it should be possible to set individual values must be clarified.

- The Provider must supply Cyberlab with a list of available identifiers (tags) together with a short description. The list should also include a summary of the restrictions applied for the individual identifiers.

7.2.2.4 Example code

N/A

7.2.3 resetExperiment

7.2.3.1 The method

public java.lang.String **resetExperiment**()

This method is optional, and is called by the Cyberlab Communication Server. Calling this method will cause the Lab Server to reset the laboratory to a predefined, ready and safe state.

Parameters

```
no parameters
```

Return value

The Lab Server returns “OK: ” represented as a String, or "ERROR: " followed by an error description.

7.2.3.2 CCS implementation details

- This method is only called by the CCS on request or based on a specific event.
- Only the Cyberlab Admin should be allowed to manually trigger this message.
- This message can be used if the Customer – Provider communication is lost during an experiment run.
- The message can also be used in order to prepare the Lab for new Users.

7.2.3.3 Lab Server implementation details

- The business logic for this method must be defined for each individual lab. Who has the right to reset a laboratory and when.

Note! It is strongly recommended that the Provider implements this method.

7.2.3.4 Example code

N/A

7.2.4 connectionIsLost

7.2.4.1 The method

public java.lang.String **connectionIsLost**(java.lang.String reason)

This method is optional, and is called by the CCS. Calling this method will send a signal to the Lab Server indicating that the communication with the Provider Lab Server is lost.

Parameters

`reason`	The reason for the lost connection

Return value

The Lab Server returns “OK: ” represented as a String, or "ERROR: " followed by an error description.

7.2.4.2 CCS implementation details

- ♦ This method is called by the CCS on request and based on some specified events.
- ♦ Only the Cyberlab Admin should be allowed to manually trigger this message.

7.2.4.3 Lab Server implementation details

- ♦ The business logic for the actions to be taken must be defined.

7.2.4.4 Example code

N/A

7.3 Not used methods

These methods are not described in any detail.

8 Cyberlab Interface - Implementation details

The Cyberlab Interface is part of the Cyberlab Communication Interface. The Cyberlab Interface is defined via a Java Interface called CyberlabInterface. The Providers must provide an implementation of this Java interface.

These methods in the CyberlabInterface are initiated by the Lab Server and sent to the Cyberlab Communication Server (CCS).

A summary of the methods is given in the table below. Some implementation details and guidelines are given in the following sections.

Table 5 CyberlabInterface – summary of methods

Method initiated by Lab Server	
Compulsory	
connect()	
loginResult(labName,sessionID,result)	
isAlive()	
Optional	
getVer() Next release: getCyberlabVersion ()	
disconnect(offlineStatusDescription)	Strongly reccomended to implement this
log(type, sessionId, action)	Strongly reccomended to implement this
checkUserAccess(sessionId)	
Not used	
getBookingData()	

Please refer to the example Java code (ProviderMain.java) for examples of calling the CyberlabInterface methods from the Provider SW.

Important files for the Provider

- The CyberlabInterface:
  filesForProvider\com\cyberlab\CyberlabInterface.java
- Example of a Provider implementation when calling the methods defined in the CyberlabInterface:
  filesForProviders\code\com\cyberlab\dummyProvider\ProviderMain.java

8.1 Compulsory methods

8.1.1 connect

8.1.1.1 The method

public java.lang.String **connect**()

This method is compulsory, and is called by the Lab Server. A call to this method will establish the connection with the CCS. The CCS returns a String giving the result of the connection.

Parameters

`no parameters`

Return value

The Cyberlab Communication Server returns "OK" in a String in order to signal that the connection was established, otherwise "ERROR: " followed by an error description.

8.1.1.2 CCS implementation details

N/A

8.1.1.3 Lab Server implementation details

- The Provider does not need to do any implementation work for this method as such.
- See details from the available example code for a simple Lab Server (dummyProvider) for details on how to implement the necessary logic around this method.

8.1.1.4 Example code

N/A

8.1.2 loginResult

8.1.2.1 The method

public java.lang.String **loginResult**(String labName, java.lang.String sessionID,
java.lang.String result)

This method is compulsory, and is called by the Lab Server. The method returns the result from attempted login (both successful and non-successful login attempts) together with the actual login time.

Parameters

`labName`	The lab identifier defined by Cyberlab
`sessionID`	The identifier for the current user (username)
`result`	The result of the attempted login Legal values are: OK: <login time> ERROR: <login time>, <error description> Login time is represented in UTC. Login time format should be YYYY-MM-DD HH:MM Typical values for the error description are: "wrong password" "not a valid timeslot" "access refused by Provider"

<NOTE> - This method is also used when a user logs off from the Lab.

<NOTE>	- The timestamp is added automatically to the message sent to the CCS. This is done in the TA level.

Return value

The Cyberlab Communication Server returns "OK" in a String or "ERROR: " followed by an error description.

8.1.2.2 CCS implementation details

The following rules apply for the CCS implementation of the *loginResult* method:

- ♦ The login result is stored in the Cyberlab DB together with a timestamp for the login.
- ♦ The timestamp is stored in UTC.
- ♦ Some providers may use the checkUserAccess method in connection with this method. Other Labs use the locally stored accessControlList in order to verify the access.

8.1.2.3 Lab Server implementation details

- ♦ The logic for verifying the username and password against the ones in the database is defined elsewhere, and may vary from Provider to Provider.
- ♦ Some providers may use the checkUserAccess method in connection with this method. Other Labs use the locally stored accessControlList in order to verify the access.

8.1.2.4 Example code

Here is a very simple example of a Provider implementation of the loginResult method. This example is from the NTNU lab using the Apis system. It also shows how the loginResult method is used both when users log in and log off.

```
   /**
    * Callback from Apis when a new user session has started.
    * The method will be called both for successful and unsuccessful login
    * attempts.
    *
    * @param nApisUserSessionID Additional identifier separating several
    *          sessions for one user.
    * @param nApisUserName Username for user logged in (the same as in the
    *          cl_session-file.
    * @param nCLientType The client type (enum: CT_NONE = 0, CT_OPC_DA = 1,
    *          CT_OPC_AE = 2). Only CT_OPC_DA is interesting for the
    *          refrigeration process.
    * @param strRole Not used. To be used in a future release of Apis with .Net
    *          role-based sequrity model.
    * @param strSessionInfo Not relevant for the Refrig. process. To be used by
    *          Apis configuration tool providing more info about the session. No
    *          of OPC groups etc.
    */
   public void OnUserSessionStarted(int nApisUserSessionID, String strUserName,
            int nClientType, String strRole, String strSessionInfo) {

     String loginString = "\"OnUserSessionStarted(): nApisUserSessionID: "
```

```
                + nApisUserSessionID
                + " strUserName: " + strUserName
                + " nClientType: " + nClientType
                + " strRole: " + strRole
                + " strSessionInfo: " + strSessionInfo + "\"";

      CyberlabInterface cyberlab = main.getCyberlabCallObject();
      ExperimentInterface callback = main.getExperimentCallObject();

      if ((cyberlab != null) && (callback != null)) {
         cyberlab.loginResult(callback.getExperimentId(), strUserName,
                "\"login: session " + nApisUserSessionID
              + " clientType " + nClientType + "\"");
      }
      else {
         if (cyberlab == null)
            // do error reporting
         if (callback == null)
            // do error reporting
      }
   }


   /**
    * Callback from Apis when a user session has ended
    *
    * @param nApisUserSessionID Additional identifier separating several
    *          sessions for one user
    * @param nReason Reason for end of session.
    */
   public void OnUserSessionEnded(int nApisUserSessionID, int nReason) {

      String logoutString = "\"OnUserSessionEnded(): nApisUserSessionID: "
              + nApisUserSessionID + "nReason: " + nReason + "\"";
      String userName = "sessionID:"+nApisUserSessionID;

      CyberlabInterface cyberlab = main.getCyberlabCallObject();
      ExperimentInterface callback = main.getExperimentCallObject();

      if ((cyberlab != null) && (callback != null)) {
         cyberlab.loginResult(callback.getExperimentId(), userName, logoutString);
      }
      else {
         if (cyberlab == null)
            // do error reporting
         if (callback == null)
            // do error reporting
      }
   }
```

8.1.3 isAlive

8.1.3.1 The method

public java.lang.String **isAlive**()

This method is compulsory, and is called by the Lab Server. It is a Cyberlab System method used internally by the system for checking the connection between the Lab Server and the Cyberlab Communication Server.

Parameters

```
no parameters
```

Return value

The Cyberlab Communication Server returns "OK" to signal that the message was received, or "ERROR: " followed by an error description.

8.1.3.2 CCS implementation details

N/A

8.1.3.3 Lab Server implementation details

- ♦ The Provider does not need to do any implementation work for this method as such.
- ♦ See details from the available example code for a simple Lab Server (dummyProvider) for details on how to implement the necessary logic around this method.
- ♦ Should be called once approx. every 1-5min from the Provider SW.

8.1.3.4 Example code

N/A

8.2 Optional methods

8.2.1 getVer

8.2.1.1 The method

public java.lang.String **getVer**()

This method is optional, and is called by the Lab Server. The method will return the current and most up-to-date version numbers of the main components in the Cyberlab system.

Parameters

```
no parameters
```

Return value

The Cyberlab Communication Server returns a String with the current and most up-to-date version available of all the Cyberlab components in the system, or “ERROR: “ followed by an error description.

The format of the version String is:

<component_1>; … <component_N>;

where

<component_i>	= <component_id>, <version>
<component_id>	The short name of the SW component: CCI Cyberlab Communication Interface (CyberlabInterface and

		ExperimentInterface)
	CCS	Cyberlab Communication Server
	TA	Transportation Adapter
	AA	Application Adapter
<version>	Version number of the SW component. This is a version number defined by Cyberlab. The version mumber follows the format <majorversion>.<minorversion>	

8.2.1.2 CCS implementation details

N/A

8.2.1.3 Lab Server implementation details

- The name will probably be changed to ***getCyberlabVersion*** in the next release.

8.2.1.4 Example code

N/A

8.2.2 disconnect

8.2.2.1 The method

public java.lang.String **disconnect**(java.lang.String offlineStatusDescription)

This method is optional, and is called by the Lab Server. The method that notifies the Cyberlab server that this experiment is about to disconnect and close the connection with the Cyberlab Server.

Note! This method may become compulsory in the next release.

Parameters

`offlineStatusDescription`	A String describing the OFFLINE status. Typical values are: Maintenance, SW upgrade, etc

Return value

The Cyberlab Communication Server returns "OK" to signal that the message was received, the last answer from Cyberlab, or "ERROR: " followed by an error description.

8.2.2.2 CCS implementation details

- The Lab status will be set to OFFLINE after this message has been received.

8.2.2.3 Lab Server implementation details

- The Providers are encouraged to implement this method even though it is not compulsory.

- The `offlineStatusDescription should be accompanied with a string indicating when the Lab is expected online again.`

8.2.2.4 Example code

N/A

8.2.3 log

8.2.3.1 The method

public java.lang.String **log**(java.lang.String type, java.lang.String sessionId,
java.lang.String action)

This method is optional, and is called by the Lab Server. The method logs some specific activity on the Lab Server. The logged action can be initiated by both the user and the system.

Note! The next release may include a timestamp from the Provider Lab Server as a parameter for this method.

Parameters

`type`	Log type. Legal values: USER, SYSTEM
`sessionId`	A session ID describing the user
`action`	The action triggered by the SYSTEM or the USER

Return value

The Cyberlab Communication Server returns “OK” in a String or “ERROR: “ followed by an error description.

8.2.3.2 CCS implementation details

- The CCS stores historical data for the actions.

8.2.3.3 Lab Server implementation details

- The Provider must define and specify the actions to be stored.

8.2.3.4 Example code

N/A

8.2.4 checkUserAccess

8.2.4.1 The method

public java.lang.String **checkUserAccess**(java.lang.String sessionId)

This method is optional, and is called by the Lab Server. The method is used by a lab to check if a user is allowed to access the lab.

Parameters

`sessionId`	A session ID describing the user

Format:

sessionId = <username>;<password>

Return value:
A String describing the access rights, or "ERROR: " followed by an error description.

Note: *This method was in the "Not used" category in the previous release!*

8.2.4.2 CCS implementation details

N/A

8.2.4.3 Lab Server implementation details

♦ Not all Labs will need this method.

8.2.4.4 Example code

N/A

8.3 Not used methods

These methods are not described in any detail.

9 The Java version of the interface

9.1 Example architecture using Java RMI

This example illustrates an example solution using Java RMI for the communication part between the CCS and the Lab Server. Java RMI is very flexibility and requires little resources to implement.

To get around the problem that Java RMI does not communicate very well with other languages than Java, it has been chosen to make a middle-layer. This middle-layer or "glue" is a Java application that translates Java RMI calls into something which can be read by any programming language; TCP/IP communication with an own protocol.

This means that the Provider must implement the defined (Java) interface and include this code segment in his own Lab Server.

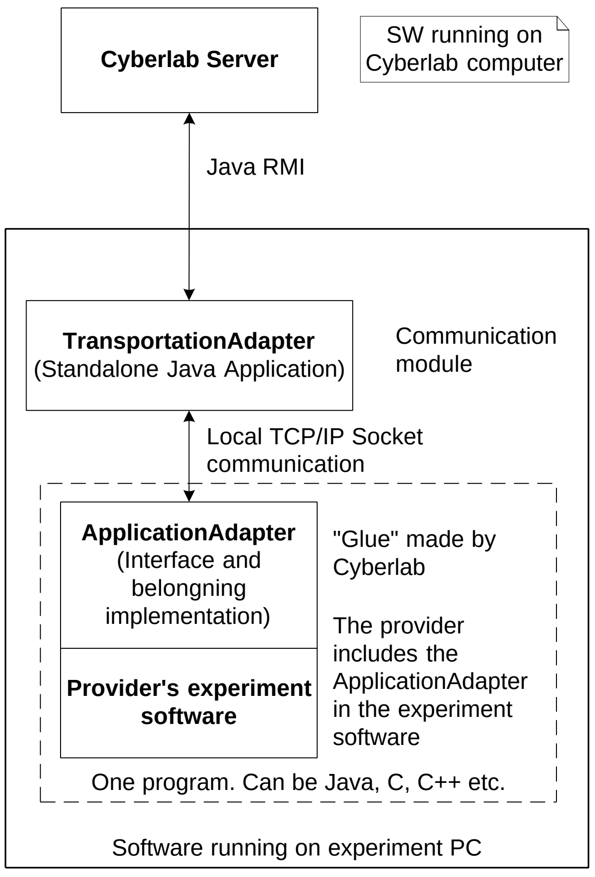

Cyberlab Server
SW running on Cyberlab computer
Java RMI
TransportationAdapter (Standalone Java Application)
Communication module
Local TCP/IP Socket communication
ApplicationAdapter (Interface and belongning implementation)
"Glue" made by Cyberlab
The provider includes the ApplicationAdapter in the experiment software
Provider's experiment software
One program. Can be Java, C, C++ etc.
Software running on experiment PC

Figure 14 Example of the architecture for the communication mechanism between the Cyberlab Server and the Lab Server using Java and the standard interface

9.2 Error recovery at the Provider side

A lot of error recovery mechanisms are built into the system in order to make it as reliable as possible. In this section, the main principle of the recovery sequence at the Provider side is given.

Before the ProviderMain calls method on the CyberlabServer, it always checks the connection by running a test method on CyberlabServer. If an error or communication problem occur, then it will try to re-establish the connection.

Situation: An error or a communication problem occurs.

Action: <u>ProviderMain in Lab Server:</u>

Start recovery sequence.

1. checkConnection() is called.
    a. isAlive() is called.
       This method will check the connection with CCS.
    b. If an error or communication problem has occurred, then isAlive() returns a string that starts with "error".
    c. stopConnection() is called.
       This method will stop the cyberlab-connection.
        i. if TransportationAdapterDispatcher and ApplicationAdapter are running, then stop it by calling:
            1. stopTAD(), which stop the TransportationAdapterDispatcher.
            2. stopApplicationAdapter(), which stop the ApplicationAdapter.
        ii. else, if only the ApplicationAdapter is running, then stop it by calling:
            1. stopApplicationAdapter(),which stop applicationadapter

2. reConnect() is called.
   This method will try to re-establish the connection.
    a. startTransportationAdapterDispatcher() is called, it starts transportationAdapter.
    b. startApplicationAdapter() is called, it starts applicationAdapter.
    c. If reConnect() could not make a connection to CyberlabServer, then it waits a specified minutes before trying again.

End recovery sequence.

10 The C++ version of the Interface

10.1.1 Introduction

A standalone Java application (the Transportation Adapter) is run on the Provider's computer. This application sets up the communication over the internet to CCS via Java/RMI. Providers connect to this Java application by either Java JNI or, as will be described here, by local sockets. Socket-communication is intended for providers using C++ projects. The package presented here is C++ code working on both Windows and Linux. There is no need for two packages. By including the class files of this package in the Provider's project, the socket communication to the Java application is handled. The Provider implements an interface and instantiates an object of a specific class. All communication to/from the CCS is then done by method calls, from the Provider's point of view.

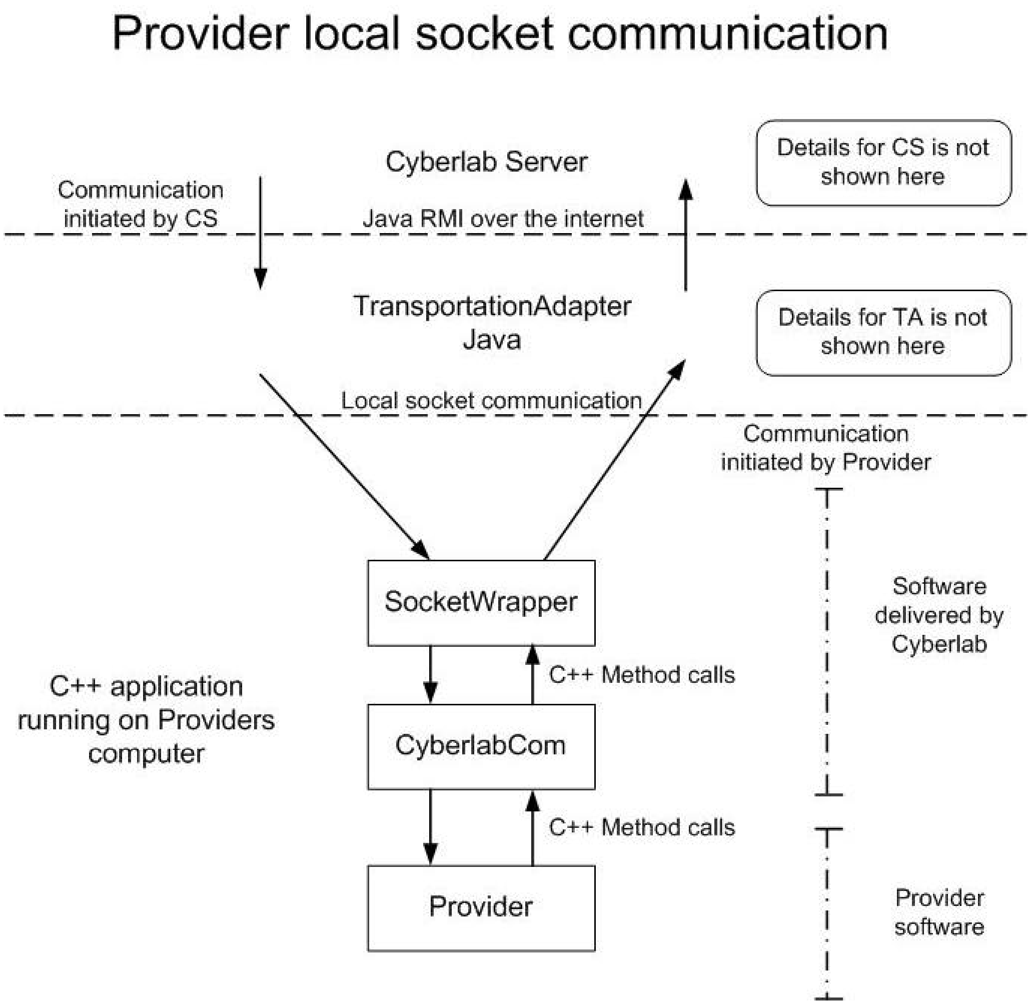

Provider local socket communication
Cyberlab Server
Details for CS is not shown here
Communication initiated by CS
Java RMI over the internet
TransportationAdapter Java
Details for TA is not shown here
Local socket communication
Communication initiated by Provider
SocketWrapper
Software delivered by Cyberlab
C++ Method calls
C++ application running on Providers computer
CyberlabCom
C++ Method calls
Provider
Provider software

Figure 15 Overview of communication between the Provider and CCS, using local sockets

10.1.2 Socket package class files

The socket package consists of three classes on Windows and four classes on Linux. The difference in numbers arise from how multithreading is handled.

Table 6 The socket package class files overview

Class files	**Description**
CyberlabCom.cpp CyberlabCom.h	Main class in the socket package. The provider makes an instance of this class. All communication with CCS goes through this class, via method calls.
SocketInterface.cpp SocketInterface.h	Cyberlab interface (virtual class). Defines all methods the Provider must provide. The same class that holds an instance of CyberlabCom in the Provider project, must also implement this interface. This interface allows CyberlabCom to call methods at the Provider.
SocketWrapper.cpp SocketWrapper.h	Manages low level socket handling, used by CyberlabCom.
ReadSocketThread.cpp ReadSocketThread.h	Used by Linux for multithreading. An instance of this class is listening on a socket for CCS calls.

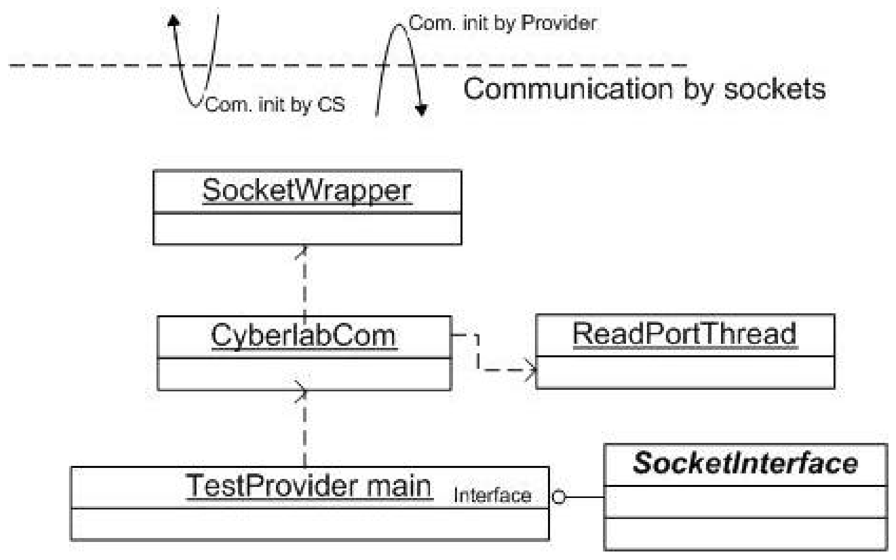


Figure 16 UML diagram of the socket package. TestProvider represents a class in the Providers project

10.1.3 Detailed description

Providers using JNI need to run a standalone Java application consisting of the Transportation-Adapter (TA) in conjunction with the ApplicationAdapter (AA). TA communicates with the CCS via RMI over the Internet. TA and AA communicate via local sockets and Providers communicate with AA through JNI. Providers using the socket package shortcut AA. The socket package communicates with TA through the same sockets AA was using. Providers using the socket package need to run TA as a standalone Java application. It is important that TA is started and running before the Providers application is started. This is to ensure correct socket creation. The TA must have permissions to start RMI- and socket communication. This is done via a Java policy-file.

The Providers projects must include the socket package. A class in the Provider project must extend SocketInterface and implement the virtual methods of SocketInterface. The same class must make an instance of CyberlabCom. CyberlabCom makes the necessary socket connections to TA, using an instance of SocketWrapper. CyberlabCom's constructor takes a pointer to the

Provider owner class which implements SocketInterface. Calls from CCS to the Provider are sent through CyberlabCom, which uses the methods in SocketInterface to invoke methods at the Provider. Calls to CCS from the Provider go through methods in CyberlabCom.

The socket package is multithreaded. The package uses two sockets for communication, one which is a client socket for communication from the provider to CCS, the other is a server socket for receiving calls from CCS. Listening on the server socket requires a thread of its own, because the thread "hangs" on the socket till something is written to it. Thread handling is different on Windows and Linux. On Windows, CyberlabCom has a static method which runs in it's own thread. On Linux the equivalent thread is defined in a class of it's own, ReadSocketThread. It is important that the Providers project supports multithreading, and includes the necessary lib-files for it. Include Ws2_32.lib in Windows, and libpthread.so in Linux.

10.1.4 C++ interface methods

Presented here are the constructors and methods available in the socket package. The Providers must take notice of the CyberlabCom class and the SocketInterface interface. The CyberlabCom class has all compulsory methods presented in the CyberlabInterface in previous sections.

Table 7 C++ version of the Java ExperimentInterface

Methods initiated by the CCS (SocketInterface Virtual Class)
Compulsory
public virtual char* getStatus()
public virtual char* getSupportedMethods ()
public virtual char* getVersion (char* identifier)
public virtual char* updateAccessControlList(char* accessList)
public virtual char* endSession()
public virtual char* getExperimentId()
Optional
public virtual char* getExperimentValue(char* identifier)
public virtual char* setExperimentValue(char* identifier, char* value)
public virtual char* resetExperiment()
public virtual char* connectionIsLost()

Note! The compulsory method getCurrentUTC is currently not implemented.

Table 8 C++ version of the Java CyberlabInterface

Methods initiated by the Lab Server (CyberlabCom Class)
Compulsory
public std::string connect()
public std::string loginResult(std::string labName, std::string sessionID, std::string result)
public std::string isAlive()
Optional
public std::string getCyberlabVersion()
public std::string disconnect(std::string offlineStatusDescription)
public std::string log(std::string type, std::string sessionID, std::string action)

Note! The optional method checkUSerAccess() is currently not implemented.

11 A simple Java Lab Server

11.1 General

The application *DummyProvider* has been implemented as a small stand-alone example program to illustrate a simple Lab Server. The dummyProvider use the ApplicationAdapter (AA) (and belonging interfaces) to communicate with the Cyberlab Communication Server (CCS) for testing purposes.

The most important issues of DummyProvider are:

- DummyProvider itself consist of only two classes: ProviderMain and CallbackClass.
- ProviderMain creates a new object from CallbackClass, and uses that object when creating a new AA. Later, the AA is used to send messages to the CCS
- CallbackClass is the class that handles all messages from the CCS.

The package com.cyberlab.dummyProvider (source code available for the Providers) contains the two test-classes for the CCS communication, and is only provided for helping the Provider getting started. It includes only a simple example of a Lab Server, but a generic Lab Server (written in Java or with a Java interface) could easily use the same methods and principals to connect to the CCS.

11.2 Summary of files

Files to run DummyProvider

- DummyProvider.jar
- CLProvider.ini (usually not needed for an Lab Server, except for the TA)
- startDummyProvider.bat

Files to rebuild DummyProvider

- com\cyberlab\*.java
- com\cyberlab\applicationAdapter\*.java
- com\cyberlab\dummyProvider\*.java
- com\cyberlab\socketCommunication\*.java
- filenamesDummyProvider
- manifestForDummyProvider
- compileDummyProvider.bat
- createJarForDummyProvider.bat

Files for JavaDoc

- All *.java files specified above (for a detailed list of all required files, see the file “filenamesDummyProvider“)
- com\cyberlab\package.html
- com\cyberlab\applicationAdapter\package.html
- com\cyberlab\dummyProvider\package.html
- com\cyberlab\socketCommunication\package.html
- createJavaDocForAll.bat (Note: Creates JavaDoc not only for DummyProvider, but for all classes and packages)

11.3 Generating DummyProvider.jar

“DummyProvider.jar” is a JAR-file which holds all classes and images needed to run DummyProvider.

The file is generated using the script “createJarForDummyProvider.bat”. Note that:

- All necessary files must be compiled (for example with “compileDummyProvider.bat“) before the JAR can be generated
- all the file names of the classes needed for DummyProvider are listed in the file “filenamesDummyProvider“
- the file “manifestForDummyProvider” is used to define the main class (which should be executed) in the JAR file.

See Sun’s web-pages for more information on how JAR files are created and used.

11.4 Configure DummyProvider

In order to run the DummyProvider, some initialisation may be required.

1. Configure CLProvider.ini
   Set the correct IP address for the computer running the CCS. Normally this will be
   CL_Server_RMI=rmi://129.241.187.53:1099/CyberlabRmiDispatcherInterface
   During testing etc, this may be set to another IP address according instructions from Cyberlab.

2. Configure CLProviderCallback.ini
   Set the correct identification for the lab using the *Experiment_Id* identifier.
   Please note that you must use a valid identifier provided from Cyberlab. Failing to use a unique and valid identifier will cause error messages trying to connect to the CCS.

   Also note that the identifier must be a valid entry in the Cyberlab DB. Please contact Cyberlab in case a new entry is needed.

3. Configure policyForExperiment
   Set the correct IP address for the computer running the CCS. This is the same IP address as for item 1. The entry will look like this:
   permission java.net.SocketPermission "129.241.187.53", "accept, connect, listen, resolve";

   Note that the policy file may have several entries.

4. Start the dummyProvider

11.5 Running DummyProvider

Windows

DummyProvider can be run by executing the file “startDummyProvider.bat” or by writing the following on the command line:

```
java -classpath
TransportationAdapter.jar;DummyProvider.jar;crypt.jar
com.cyberlab.dummyProvider.ProviderMain
```

If the ProviderMain starts the TA, and the ini-files are not located in current directory, the location of the ini-files must be specified by the *cyberlab.inifile.dir* property.

The property can be set in the bat-file starting the ProviderMain as follows:

```
java -Dcyberlab.inifile.dir=<path_to_the_inifile> \
-classpath TransportationAdapter.jar;DummyProvider.jar;crypt.jar \
com.cyberlab.dummyProvider.ProviderMain
```

When the TA is started by the dummyProvider (or whatever it is called), the TA will get the cyberlab.inifile.dir

property if it is set.

Unix/Linux

DummyProvider can be run by writing a command file (.sh or similar) or by writing the following on the command line:

```
java -classpath
TransportationAdapter.jar:DummyProvider.jar:crypt.jar
com.cyberlab.dummyProvider.ProviderMain
```

Please note the ':' separators used for Unix/Linux, and the ';' separators for Windows.

If the ini-files are not located in current directory, the location of the ini-files can be specified by the *cyberlab.inifile.dir* property as indicated in the example above for the Windows version.

The above assumes that Sun's JDK 1.3 or JRE 1.3 is installed and properly configured.

11.6 The TransportationAdapter

A Lab Server that shall communicate with the CCS must use a Java application made by Cyberlab, "TransportationAdapter" (TA), to enable the communication from the Lab Server computer to the CCS. The name TransportationAdapter implies that the application adapts the messages sent between the Lab Server and the CCS.

The current version of TA use Java RMI communication towards the CCS. It is therefore important that the Lab Server, if put behind a firewall, is made sure to be able to communicate on the default ports used by Java RMI. A description of how to use RMI behind a firewall can be found on Sun's RMI web site at the FAQ "http:// java.sun.com/j2se/1.3/docs/guide/rmi/faq.html#firewall".

No programming is needed for the Provider to use the TA, but it is important that the Provider knows that TA shall be used when doing further programming of the Lab Server. This is because the ApplicationAdapter (AA) (described below) requires a running TA in order to communicate with the CCS.

11.6.1 Running the TransportationAdapter

The TA can be started manually, or it can be started automatically by the AA. The default mode is to start the TA automatically.

The TA will need the configuration-files (CLProvider.ini and CLProviderCallback.ini) in order to operate correctly on the Provider's computer. In order to locate the ini-files, the TA will use the property *cyberlab.inifile.path* if set. Otherwise, the TA will use the standard system property *user.dir* as the path for the ini-file.

Started TA manually is only recommended in special cases, and is only recommended for experienced users. If it is started manually, it must be done before starting the AA via the Lab Server. However, manually starting TA will not influence the behaviour of the system.

The TA can be run by executing the file “startTransportationAdapter.bat” or by writing the following on the command line:

```
java -Djava.secutiry.manager \
     -Djava.security.policy=policyForExperiment \
     -jar TransportationAdapter.jar
```

If the location of the ini-files are not the current directory, the location of the ini-files may be specified in a similar way as when starting the dummyProvider in the previous section:

```
java -Dcyberlab.inifile.dir=<path_to_the_inifile> \
     -Djava.secutiry.manager \
     -Djava.security.policy=policyForExperiment \
     -jar TransportationAdapter.jar -d
```

The above also assumes that Sun’s JDK 1.3 or JRE 1.3 is installed and properly configured. TransportationAdapter can be given the following parameters (which must be written at the very end of the line when starting TA):

Table 9 Program arguments for TransportationAdapter

Parameter	Description
-d	Print debug-information to default out
-pXXXX	Listen on port1 = XXXX when communicating with the experiment software (default is 5678 or the value in CLProvider.ini)
-qXXXX	Listen on port2 = XXXX when communicating with the experiment software (default is 5688 or the value in CLProvider.ini)

Command line arguments override any settings in the INI-file "CLProvider.ini".

11.7 Classes and Interfaces provided by Cyberlab

Cyberlab provides the TA, the AA and the interfaces *ExperimentInterface* and *CyberlabInterface* for the Provider to use.

The source files can be found at “filesForProviders\com\*\”, and precompiled versions can be found in “filesForProviders\ApplicationAdapter.jar“. The files can be compile into the Lab Server, or simply put in the classpath.

11.7.1 Interface com.cyberlab.ExperimentInterface

This is the interface that defines all the method calls which the Cyberlab Server can call on an Lab Server. See the JavaDoc for a more detailed description of the "public interface ExperimentInterface” interface and its methods.

The Provider must create a class that implements ExperimentInterface. The class must be instantiated before any communication with the Cyberlab Server is started, and calls to the object are calls from the Cyberlab Server. For example, as done with DummyProvider, a class named CallbackClass can be created like this:

```
package something;
import com.cyberlab.ExperimentInterface;
```

```
public class CallbackClass implements ExperimentInterface {
    private static final String EXPERIMENT_ID = "DummyExperiment123";

public CallbackClass() {
    doSomeInitializationIfNessesary();
}
public String getExperimentId() {
    return EXPERIMENT_ID;
}
public String getExperimentValue(String identifier) {
    doSomething();
    return someString;
}
public String setExperimentValue(String identifier, double value) {
    doSomething();
    return someString;
}
public String resetExperiment() {
    doSomething();
    return someString;
}
public String throwOutUser(String sessionId) {
    doSomething();
    return someString;
}
.... the rest of the methods are put here...
}
```

11.7.2 Interface com.cyberlab.CyberlabInterface

This is the interface which defines all the method calls which a Lab Server can call on the Cyberlab Communication Sever. See the JavaDoc for a more detailed description of the interface and its methods.

11.7.3 Class com.cyberlab.applicationAdapter.ApplicationAdapter

The ApplicationAdapter (AA) is the class which automatically sets up communication to TransportationAdapter (TA) (and thus the CCS). The AA has a constructor which takes a object (any class implementing ExperimentInterface, the CallbackClass) as the only parameter. That object is used when the AA receives calls from the CCS.

When creating a new instance of AA, the following (or similar) code must therefore be used:

```
...
// Connects to the Cyberlab Server
ExperimentInterface callback = new CallbackClass();
try {
    cyberlab = new ApplicationAdapter(callback);
}
catch (UnknownHostException e) {
    System.out.println("Error: Can not find host!!!\n" + e + "\nExiting!");
    System.exit(-1);
}
catch (IOException e) {
    System.out.println("Error: IO-error!!!\n" + e + "\nExiting!");
    System.exit(-1);
}
...
```

The above example is taken from DummyProvider. It assumes that the variable “cyberlab” is defined somewhere above the code shown.

After the variable “cyberlab” has been initialised as shown in the code above, method calls can be sent to the CCS. Some code examples are shown below. See the example code in ProviderMain.java for more examples:

// Example 1: Get version

```
if(checkConnection() == false)     // Check connection with CCS
{
  reConnect();                     // Re-establish connection with CCS
  System.out.println("The version is: " + cyberlab.getVer());
}
else
{
  System.out.println("The version is: " + cyberlab.getVer());
}
```

// Example 2: Disconnect from the CCS

```
if(checkConnection() == false)
{
  reConnect();
  System.out.println("I say 'disconnect', the answer is \"" +
       cyberlab.disconnect("Shutdown due to scheduled upgrade") + "\"");

  stopConnection();                // Stop CCS-connection
}
else
{
  System.out.println("I say 'disconnect', the answer is \"" +
       cyberlab.disconnect("Shutdown due to scheduled upgrade") + "\"");

  stopConnection();                // Stop CCS-connection
}
```

12 Configuration files for the Provider

12.1 General

The following configuration files must be set for the Lab Server to communicate with the Cyberlab Communication Server:

CLProvider.ini
CLProviderCallback.ini
policyForExperiment

12.2 CLProvider.ini

The “CLProvider.ini” file holds the main settings for the TransportationAdapter. The file should be located in the same directory as the directory where the Provider SW (TransportationAdapter) is started from.

A typical “CLProvider.ini” file can look like this:

```
# Transportation Adapter and Application Adapter properties
# Used for both Transportation Adapter application
# and Provider SW
# The last line must be empty!!!

# Port definitions (Used by both TA and AA)
TA_Port1=5678
TA_Port2=5688

# Cyberlab server RMI connection (used by TA)
# 129.241.10.126  kybpc-bb8
# 129.241.187.53  Cyberlab server (linux)
# CL_Server_RMI=rmi://localhost:1099/CyberlabRmiDispatcherInterface
CL_Server_RMI=rmi://cyberlab.org:1099/CyberlabRmiDispatcherInterface
```

The syntax of the INI-file is simple: All lines starting with # are comments. All empty lines are ignored. All other lines have a property name, equals sign “=” and the value of the property.

Note that the last line must be blank, or else the last property will not be read properly!

The properties available for use in CLProvider.ini are shown in the table below:

Table 10 Properties in CLProvider.ini

Property name	Description	Example value
TA_Port1	The first port to listen for AA connection. Used for communication between TA and AA.	5678
TA_Port2	The second port to listen for AA connection. Used for communication between TA and AA.	5688
CL_Server_RMI	Address to Cyberlab Server RMI service	rmi://cyberlab.org:1099/ CyberlabRmiDispatcherInterface rmi://localhost:1099/CyberlabRmiDispatcherInterface

If some of the properties are missing (or the whole INI-file is missing), hard-coded (in the Java code) default values will be used for the missing values.

12.3 CLProviderCallback.ini

The “CLProviderCallback.ini” file holds the main settings for the Provider Lab Server callback properties. The file should be located in the same directory as the directory that the Provider Lab Server is started from. A typical “CLProviderCallback.ini” file can look like this:

```
# Cyberlab Provider Lab Server Callback properties
# Comments start with #
# The last line must be empty!!!

#==================================================
# The experiment ID of this experiment
# It is sent to Cyberlab when getExperimentId()
# is called. The ID is the same as the Laboratory
# name on the Cyberlab Web
#
# The syntax is
# Experiment_Id=<lab name>
#
#
# The following IDs (<lab name>) should be used
# by the ReLAX partners:
# -----------------------------------------------
# Experiment_Id=Refrigeration cycle process
# Experiment_Id=Head box
# Experiment_Id=The Inverted Pendulum
# Experiment_Id=The Electrical Drive
# Experiment_Id=Optical Tracking System
#
# For testing, the following IDs can be used:
#------------------------------------------------
# Experiment_Id=DummyLabRUB
# Experiment_Id=DummyLabEPFL
# Experiment_Id=DummyLabNTNU
#==================================================

# dummy experiment. To be replaced by the Provider
Experiment_Id=DummyLab123

# Description of the current SW version for
# the specified SW components
# TA_VERSION: TransportationAdapter Version
# AA_Version: ApplicationAdapter Version
# ES_Version: ExperimentServer Version
#

# Name of the file used by the Provider to
# store the upcomming sessions sent from the
# Cyberlab Communication Server
#
Filename=ProviderSessionFile.txt
```

The syntax of the INI-file is simple: All lines starting with # are comments. All empty lines are ignored. All other lines have a property name, equals sign “=” and the value of the property.

Note that the last line must be blank, or else the last property will not be read properly!

The properties available for use in CLProviderCallback.ini are shown in the table below:

Table 11 Properties in CLProviderCallback.ini

Property name	Description	Example value
Experiment_Id	Identifier of this lab. Will be sent to the Cyberlab system.	DummyLab123 Refrigeration cycle process Head box The Inverted Pendulum The Electrical Drive Optical Tracking System
Filename	Name of the file used by the Provider to store the updated access control list	ProviderSessionFile.txt

12.4 Policy file – policyForExperiment

On some systems TransportationAdapter is not given the correct default security access rights. Therefore the following policy file is used when starting the application:

```
/* AUTOMATICALLY GENERATED ON Wed Jan 10 11:32:07 GMT+01:00 2001*/
/* DO NOT EDIT */

grant codeBase "file:-" {
  permission java.net.SocketPermission "localhost", "accept,
          connect, listen, resolve";
  permission java.net.SocketPermission "129.241.187.64", "accept,
          connect, listen, resolve";
  permission java.net.SocketPermission "cyberlab.org", "accept,
          resolve";
  permission java.io.FilePermission "<<ALL FILES>>", "read, write";
  permission java.util.PropertyPermission "user.dir", "read";
  permission java.util.PropertyPermission "cyberlab.inifile.dir", "read,
write";
};
```

Important note:

- The policy-file must include the IP address of the machine running the Transportation Adapter.
- The user.dir is read by the Ta for locating the ini-files.
- The cyberlab.inifile.dir property must have both read and write access. This is used by the TA for the location of the ini-files.
- On the Refirgeration Process at NTNU, the policy-file does only work if starting with ‘grant {‘ instead of ‘grant codeBase "file:-" {‘

It is recommended that the policy file is only created or changed using the JDK tool “policytool”. The tool can be started by writing “policytool” on the command line. See Sun’s web pages for a more detailed description.

13 Access Control

This section describes one of the possible access control schemes to be applied for Lab Users booking time slots and running the Labs connected to the Cyberlab System.

13.1 Password encryption algorithms

The SUN's Java crypter classes of the JCE package are only running up from Java 1.3.

Some of the providers are using Java 1.1, and we will therefore use a small a small crypter package 'Crypto'. This package will also work for Java 1.0.

The Crypto package has an encryption and a decryption method.

13.1.1 Example of using the Crypto package

The Crypto package can be tested using a simple test program CryTest e.g. by entering

```
java -classpath .;Crypt.jar CryTest "I want to encrypt this and see it decrypted again"
```

For the usage in the Cyberlab system, the username and the password are encrypted in one single string by separating the username from the password by a '|'.

The encryption can be simply performed by calling

```
String userID = crypter.encryptString(UserName+"|"+Password);
```

This string can then be used for the user ID throughout the Cyberlab system.

The original information can be later extracted by

```
String UserNameAndPassword = crypter.decryptString(userID);
```

and then simply split into UserName and Password at the '|'.

13.2 User registration and User login at Provider Lab using the encryption scheme

This is the normal procedure to be applied for the Labs.

1. The user register as a Lab User at the Cyberlab Portal site, and requests Customer status.
2. The user may also have been registered by someone else via the Bulk booking scheme. If so, the user will receive the username and password via email from Cyberlab.
3. The user can change the password whenever he/she wants.
4. The user makes bookings for a specific lab.
5. The bookings are sent to the Provider's Lab Server together with the username and password in encrypted form.
6. If needed, the Provider will manually create a local user using the provided username and password.
7. The encrypted password is stored in a local Provider DB together with the username.
8. The User will logon to the Provider Lab using his Cyberlab Portal username and password.
9. The Provider will call a method VerifyUser(username, password) returning true/false.
10. A message *loginResult* is sent from the Provider to the Cyberlab Server.
11. If login is accepted, the Lab application is started and operation is made available for the user, and
12. the Cyberlab Status Window is launched on the User's computer.

Cyberlab will require an agreement with the Provider concerning protection of the password.

13.3 Provider with manual registration procedure

1. The user registers as a new user at the Cyberlab Portal site.
2. The user selects the username and password for the Cyberlab Portal.
   However, in some situations like bulk booking, the Cyberlab System will automatically generate a username.
3. Cyberlab stores the username and password (encrypted) in the Cyberlab DB.
4. When the new user books a lab session on a lab from a specific Provider,
5. Cyberlab registers the User as a “non-confirmed” user for Labs from the specific Provider.
6. In this case, only bookings more than X working days from will be accepted. This in order to allow for manual local user registration at the Provider site.
7. For this “first time user”, Cyberlab sends a user registration request by email to the Provider Admin informing about:
   Username, full name, email
8. The Provider registers the new user and
9. The Provider sends an email to Cyberlab notifying the result of the registration.
10. Cyberlab will send the new user a notification email if the registration process at the Provider site failed for some reason.
11. Cyberlab registers the User as a “confirmed” user for the Labs from this specific Provider.
12. A confirmation email will automatically be sent from Cyberlab to the User including information about:
    - the booked timeslots
    – instructions on how to change / cancel bookings
    – specific Provider login instructions
    – the username to use for logging into the Lab (== Cyberlab username), but no password
13. The Provider sends an email with the password to the User. This password is only valid for the Labs from this specific Provider.
14. The message *updateAccessControlList* containing upcoming lab sessions is sent from Cyberlab to the Provider.
15. The Provider stores the updated access control list in a protected DB or file on the Provider computer.
16. The User can log into the Lab from a Provider web-page using the same username as for the Cyberlab site, but with a Provider specific password.
17. A message *loginResult* is sent from the Provider to the Cyberlab Server
18. If login is accepted, the Lab application is started and operation is made available for the user, and
19. the Cyberlab Status Window is launched on the User’s computer.

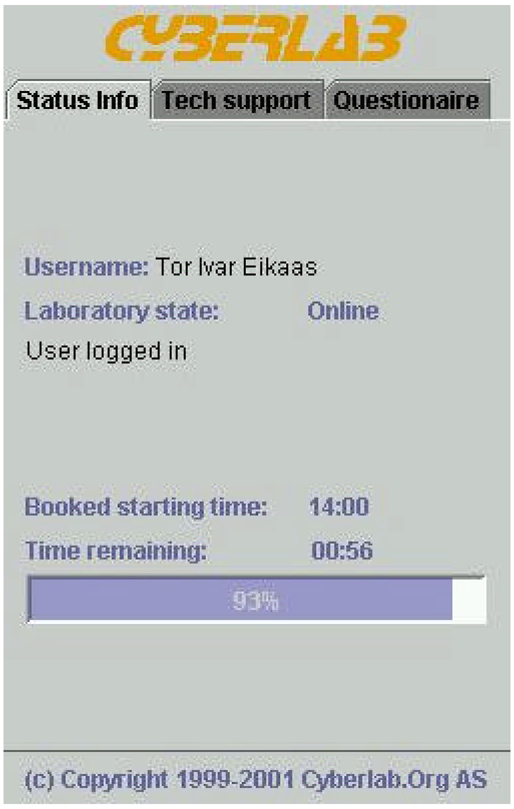
14 Run-time information to Lab Users

14.1 General

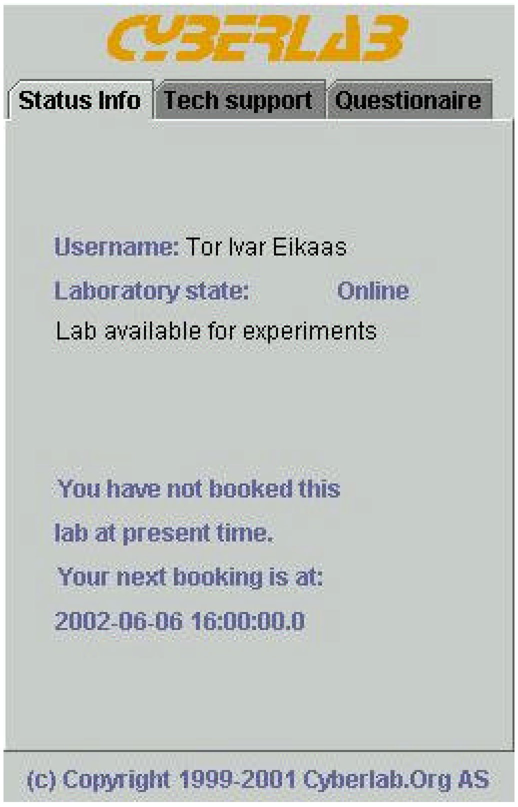
When a Lab User is trying to log into a lab and when he is running some experimentation on a lab, is sometimes a need for additional information to be displayed for the Lab User. This can for example be general information about the current session he has booked, status information from the Lab, help information, request for user feedback etc.

In order to improve the total quality of the system (of both the Cyberlab Portal and the Lab application), Cyberlab has developed a Cyberlab Information Applet that can be configured for each individual lab.

Typical information displayed in the prototype version of the applet is given in the two following figures below.



Username: Tor Ivar Eikaas
Laboratory state: Online
User logged in
Booked starting time: 14:00
Time remaining: 00:56
93%
(c) Copyright 1999-2001 Cyberlab.Org AS

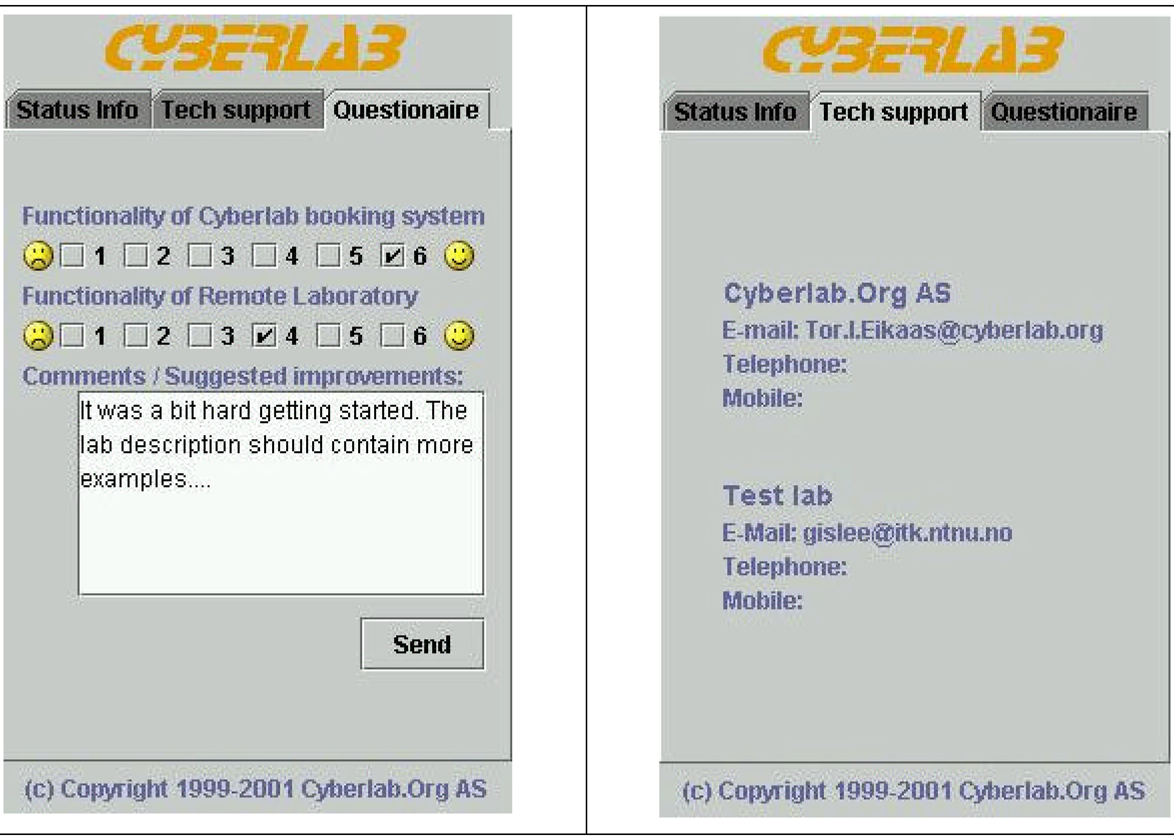


Username: Tor Ivar Eikaas
Laboratory state: Online
Lab available for experiments
You have not booked this
lab at present time.
Your next booking is at:
2002-06-06 16:00:00.0
(c) Copyright 1999-2001 Cyberlab.Org AS

Figure 17 Cyberlab Information applet after a successful login and after a non-successful one.

In Figure 17, the applet to the left shows the typical information after a successful login to the Lab. The applet shows the remaining time of the booked timeslot(s) together with important status information from the lab. This status information can be useful for the Lab User if the connection between the Lab User client application and the lab breaks down, or if there are unforeseen problems with the Lab User client application.

The applet to the right shows the typical information after a non-successful login attempt. The applet will then show a list with the next few booked and registered timeslots for this user at this specific lab. The applet will then be useful for the Lab User in order to know how to proceed further. One typical reason for a non-successful login could be problems with the local timezone offset to UTC.



Status Info | Tech support | Questionaire

Functionality of Cyberlab booking system

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☑ 6

Functionality of Remote Laboratory

☐ 1 ☐ 2 ☐ 3 ☑ 4 ☐ 5 ☐ 6

Comments / Suggested improvements:

It was a bit hard getting started. The lab description should contain more examples....

Send

(c) Copyright 1999-2001 Cyberlab.Org AS

Status Info | Tech support | Questionaire

Cyberlab.Org AS
E-mail: Tor.I.Eikaas@cyberlab.org
Telephone:
Mobile:

Test lab
E-Mail: gislee@itk.ntnu.no
Telephone:
Mobile:

(c) Copyright 1999-2001 Cyberlab.Org AS

Figure 18 Questionnaire and Tech support from the Cyberlab Information Applet

In case of problems, the user can get some links to both administrative and technical help via the “Tech support” part of the applet. This is shown to the right in Figure 18.

The Lab Users are also encouraged to rate the lab and the Cyberlab System after they have run the lab. This information is important feedback for both the ESP and the Lab owner. In addition, potential Customers can use statistics about the individual labs in order to select the best ones (most reliable, best functionality etc).

14.2 Integrating the Cyberlab Information Window with the Provider SW

Please contact Cyberlab in order to get details on how to integrate the Cyberlab Information Window with the Provider SW.