

## Creating A New Experiment GUI

Version 1.0

# Creating A New Experiment GUI

## Overview

Remote Experienced Control (ReC) is a data acquisition and remote control platform through which any student or teacher can perform real experiments in virtual environments.

ReC is composed by three main architectural blocks: ReC Hardware Server, ReC Multicast Server and ReC Client.



This guide will focus on the creation of experiment plug-in modules to provide on the ReC Client application.

The experiment's graphical user interface provides an easy way for the elab user to configure the experiment execution parameters and watch the results as they are being produced in quasi-realtime.

## New Experiment GUI

The experiment's graphical user interface provides an easy way for the elab user to configure the experiment execution parameters and watch the results as they are being produced in quasi-realtime.

## The Experiment Container

ReC NewFace UI acts as a base container for all experiments. As such, each new experiment is plugged in into this base user interface and interacts with it. So, to be able to plug into this base user interface, the new interface components for each experiment are required to implement some Java interfaces that enable it to bootstrap onto the framework code. Obviously, the lifecycle of each user interface component is defined by the underlying base UI implementation

## Goal

This document aims to provide information to guide developers in the process of creating the user interface components required by the ReC Client application in order to add a new experiment as a plug-in module.

## Intended audience

This document is targeted at software developers that need to create a graphical user interface to interact with experiments within the ReC Client application.

## Expected background knowledge

Solid Java programming language  
Swing API theory and practice  
Model View Controller pattern

## Getting Started

In this chapter, we will cover the steps to create a new Experiment GUI. As an example we used the experiment "Toilet Paper" (this is only a [gedankenexperiment](#) ).

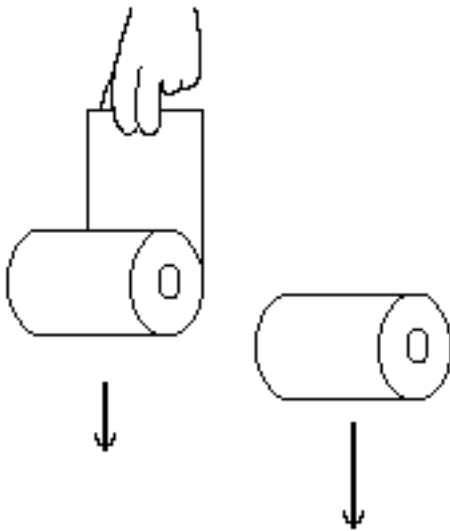
The goal is to illustrate the effect of rotation of a body in its total time of free fall.

### Experiment Description

Drop two rolls of toilet paper of the same size. One is completely free to fall and the other hung by a tip.

Of course, the free roll reaches the ground first because the acceleration of the first roll is less stuck.

The goal of the experiment is to predict the initial heights that the rollers must have in order to reach the ground simultaneously. In the process, you have to use concepts of acceleration and momentum of inertia.



## Step I - Creating the structure for a new experiment

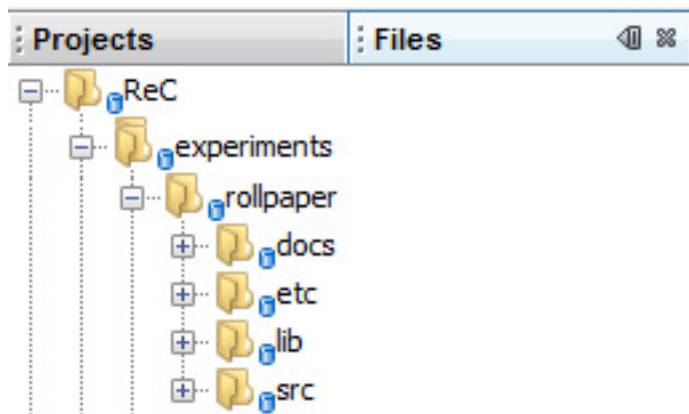
Create a new project for the experiment, in \$REC\_HOME/experiments using the following command:

```
ant -f build_new_experiment.xml create.experiment -Dexperiment.name =rollpaper
```

in \$REC\_HOME/templates/ (there will be an ant build file called build\_new\_experiment.xml).

The name of the experiment should be small, such as an acronym, without any spaces or special characters such as 'ç', 'ã', etc.

After the script is run, the following structure should have been created:



An explanation of this structure follows (each folder should have a small readme text file with a short description of its purpose):

- docs - directory containing general documentation related with the experiment, such as protocols (originals from producers), apparatus descriptions, specific datasheets, etc.
  - CommonDatasheets - datasheets related to components and devices that could be used in more than one experiment. If the datasheet is related only to a particular experiment, it should be moved to Apparatus\Docs.
  - e-escola - original doc/html or txt documents from e-escola topics.
    - examples - reports and data analysis of each experiments. It can be .xls, .doc, .pdf, etc, but open formats are preferential.
    - presentation - presentations related to the experiment's subject; it should contain original material to be treated by teachers as supporting material for lessons.
  - microcontroller - documentation and zip source files for the microcontroller.
    - codesources - repository of .zip files containing all source and libs necessary for compiling and running the code on the microcontroller.
    - PCB - specific designs of printed circuit boards and plugs/connectors for each experiment.
  - pictures - pictures and schematics sources (like .fla or Java scips) from each experiment.
- etc - it should contain one build.xml to override specific experiment build configurations, as well as its corresponding build.properties file. It also contains the ReCBaseUIConfig.xml (deprecated) and ReCFaceConfig.xml GUI configuration files, whose contents will be explained later.
- lib
  - java - in this directory, you should place the jar files that should be available for both server and client parts of this specific experiment, not already included in the base libraries for ReC or e-lab base structures.
    - client - in this directory, you should place the jar files that should be available ONLY for client parts of this specific experiment, not already included in the base libraries for ReC or e-lab base structures.
    - server - in this directory, you should place the jar files that should be available ONLY for server parts of this specific experiment, not already included in the base libraries for ReC or e-lab base structures.
- src
  - java
    - client - the Customizer and Displays code for this experiment. You may extend the platform base code both from e-lab and ReC as needed, but the sources should be located here for build system correct integration.
    - server - the driver's code for this experiment. You may extend the platform base code both from e-lab and ReC as needed, but the sources should be located here for build system correct integration.
  - pic - in this directory, you should place the source files for the PIC code for this experiment controller.
  - stamp - in this directory you should put the source files for the basic stamp 2 code for this experiment controller.

Notes:

- The previous structure should not be changed or the deployment system will stop working
- If there is already an experiment with the same name, nothing will be created/deleted/overwritten.

Additionally, the previous ant target updates the file `$REC_HOME/templates/build.experiments.properties`, adding the created experiment to its list. This file should not be changed manually.

## Step 2 - Creating the Customizer

A Customizer is a graphical component that handles the user input for each configuration parameter in order to define an experiment environment. Each experiment has one single Customizer to enable the definition of valid configuration values according to the experiment specification.

To be able to design the user interface, some questions should be answered in advance:

- What are the experiment parameters?
  - Define each parameter's name, type, range, and default value.
- What are the acquisition frequencies and sampling scales?
- What are the ranges of number of samples?

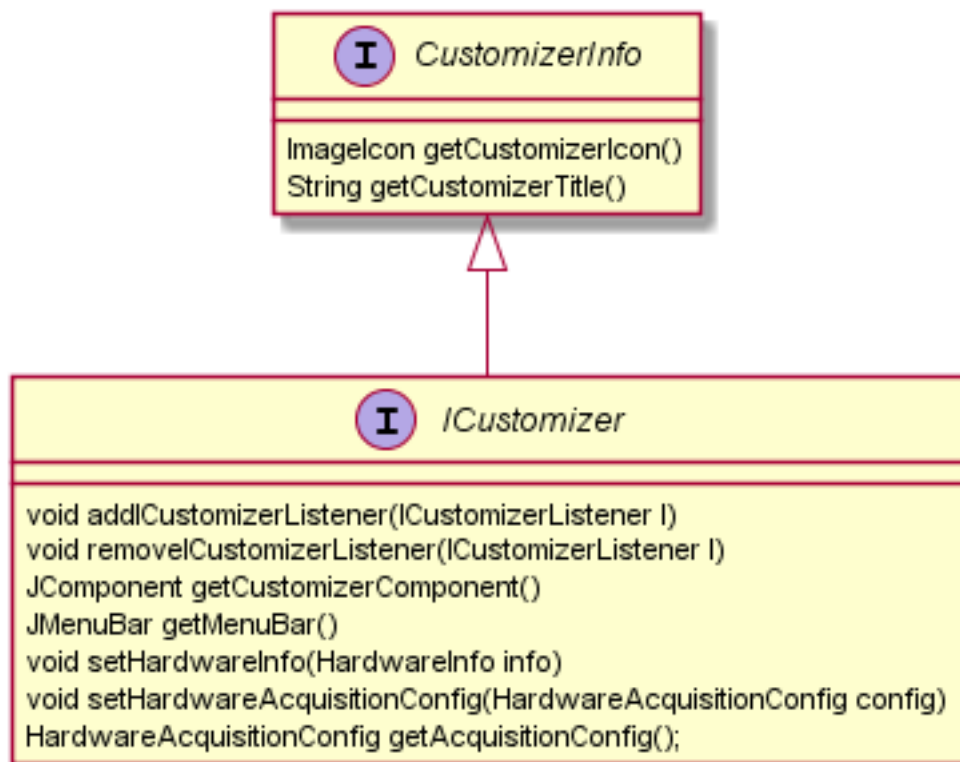
The answers to these questions will define the user input needed to set the acquisition configuration. There is a template of a spreadsheet that could be used to fill this information at : <https://svn.linkare.com/internal/rec-project/rec/trunk/docs/rec>

### Implementing the Customizer

The Customizer implementation should be placed in `$experiment_name/src/java/client/<full_package_to_experiment>`. E.g.,

`$REC_HOME/experiments/rollpaper/src/java/client/com/mycompany/elab/client/rollpaper`

The `com.linkare.rec.impl.client.customizer.ICustomizer` java interface defines the required implementation to be able to correctly set up the graphical user interface for an apparatus and to inform listeners that the acquisition configuration is OK to be used and has been customized to the user's particular needs.



In the package `$REC_HOME/experiments/rollpaper/src/java/client/pt/utl/ist/elab/client/rollpaper`, create a new java class called `RollPaperCustomizer`, this class will be responsible for implementing the methods needed to create the customized.

This class could extend `JComponent` either directly or indirectly, this case will extend from `JPanel`, this is not mandatory, it can implement a class with the components which find it necessary, and after to reuse in the `RollPaperCustomizer`, how best practice, we recommend use the `JPanel`, this class must also implement the `ICustomizer` of the package `com.linkare.rec.impl.client.customizer`.

In addition to the methods defined by `ICustomizer` (and `CustomizerInfo`), you must have a public no-argument constructor, because the base platform requires such a constructor to dynamically and reflectively instantiate your customizer.

```

public RollPaperCustomizer () {
    //instantiation code here...
}
  
```

This constructor should not throw exceptions and should not acquire any resources, because it will be displayed in the User Interface and there are other methods better for the initialization code. Any exception should be handled inside the component itself.

## Major Customizer Methods to implement

The base UI container of ReC will call the following methods of the Customizer in this order (lifecycle is guaranteed):

### `addICustomizerListener`

This method is the way the base UI container registers himself as an observer for the Customizer's state changes. The customizer should keep an internal reference to the `ICustomizerListener` references passed onto it and send them the appropriate events:

- **DONE** - when the user chooses a valid configuration and is ready to run the experiment, enabling the Play button;

- CANCELED - when the user chooses an invalid configuration - which should result in disabling the Play Button on the interface;

One of the many possible ways to implement this code is:

```
public synchronized void addICustomizerListener ( final ICustomizerListener listener ) {
    if (listenerList == null ) {
        listenerList = new javax . swing . event . EventListenerList ();
    }
    listenerList.add (ICustomizerListener . class , listener );
}
```

#### setHardwareInfo

This method is called to pass the meta-information of the hardware, such as outputs, inputs, frequency scales, sample scales, etc. of the experiment. Whichever UI components were created in the initialization of the interface could now be adjusted to match the hardware specified ranges.

Example:

```
@Override
public void setHardwareInfo ( final HardwareInfo hardwareInfo ) {
    this . hardwareInfo = hardwareInfo ;
    ChannelParameter hiParam = this . hardwareInfo . getHardwareParameter ( "hl" );
    sliderHeigthFallOne.setMaximum (Integer . valueOf (hiParam . getParameterSelectionList ()
[ MAXIMUM_VALUE_INDEX ]));
    sliderHeigthFallOne.setMinimum (Integer . valueOf (hiParam . getParameterSelectionList ()
[ MINIMUM_VALUE_INDEX ]));
    sliderHeigthFallOne.setMinorTickSpacing (Integer . valueOf (hiParam .
getParameterSelectionList ()[ STEP_VALUE_INDEX ]));
    sliderHeigthFallOne.setSnapToTicks ( true );
    //... more code after this ...
}
```

#### setHardwareAcquisitionConfig

This method is called to pass into the Customizer a reference for the HardwareAcquisitionConfig currently set in the hardware (The last experiment that was run). This method could update the Customizer interface components as required to show the user the last valid configuration that any experiment was run with.

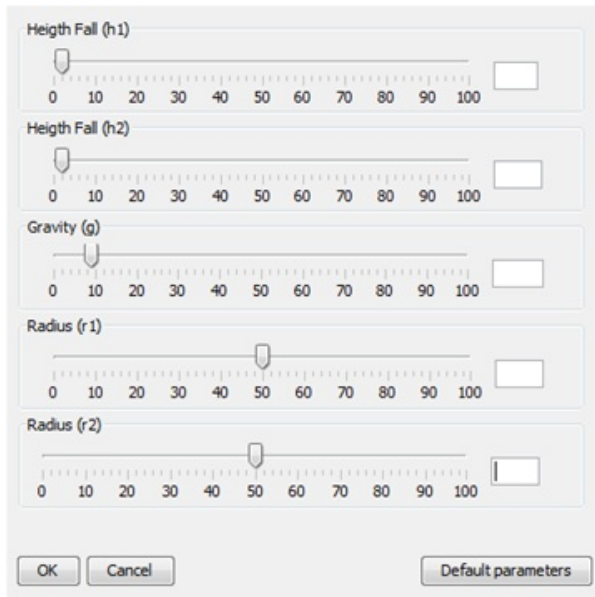
A specific customizer could potentially ignore this method and show a default configuration of it's choice, or rather update the interface to show the client the latest configuration made by the last experiment's user.

```
@Override
public void setHardwareAcquisitionConfig ( final HardwareAcquisitionConfig acqConfig ) {
    this . acqConfig = acqConfig ;
    String hICurrentSelectedValue = acqConfig . getSelectedHardwareParameterValue ( "hl" );
    heigthFallValue1.setText (hICurrentSelectedValue );
    sliderHeigthFallOne.setValue (Integer . valueOf (heigthFallValue1 . getText ());
    //... more code after this ...
}
```

#### getCustomizerComponent

This method should return the visual components to be shown to the user. As such, the Customizer could extend from javax.swing.JComponent and return a reference to this from this method, or it may choose to delegate the UI interaction to a different class and return a reference to this new component. Please note that this visual component should not be instantiated several times, as this method may be called several times in one single access to the experiment.

As an example for the described experiment, it was necessary to create the following graphical interface:



The parameters of this experiment are: Height Fall(h1), Height Fall(h2), Gravity (g), Radius(r1) e Radius(r2).

In this case as the RollPaperCustomizer both extends from javax.swing.JPanel and implements the com.linkare.rec.impl.client.customizer.ICustomizer, this method just returns this object's reference.

```
public JComponent getCustomizerComponent () {
    return this ;
}
```

## Step 3 - Creating Displays

The displays, placed in <experiment\_name>/src/java/client/<full\_package\_to\_experiment>. E.g.,

\$REC\_HOME/experiments/rollpaper/src/java/client/pt/utl/ist/elab/client/rollpaper

are the components for presenting the experiment's results. Those results may be presented as

- Graphics
- Tables
- Videos
- Any specific gauge, sensor or other display forms (could even play a sound, for instance)

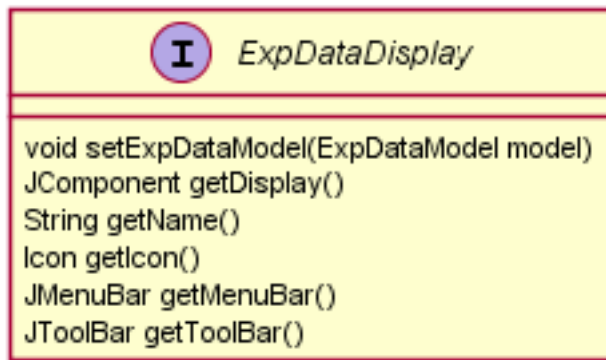
To get some basic understanding about the experiment's results, some questions must be answered:

- What are the experiment's data input channels?
  - Define the results channel name, measurement unit, scale, range and error
- How can the results be correlated to each other
  - Get a description of the experiment and the physical laws being demonstrated by the experiment

## The Display Interface

The com.linkare.rec.impl.client.experiment.ExpDataDisplay Java interface defines the methods needed by the framework to be implemented in the displays.





## Default Displays

To simplify the displays development some default implementations are provided on the framework, both at ReC and eLab levels.

### Data Table

Usually, this is the first display to be created in order to show the experiment results in a data table. To do this, just extend the `com.linkare.rec.impl.baseUI.table.DefaultExperimentDataTable` on your package, like:

```

package pt . utl . ist . elab . client . g ;
public class GDataTable extends com . linkare . rec . impl . baseUI . table . DefaultExperimentDataTable
{
    public GDataTable () {
        super ();
        final pt . utl . ist . elab . client . g . GTableModelProxy model = new pt . utl . ist . elab . client . g .
        GTableModelProxy ();
        setActualTableModel(model );
        setExpDataModelContainer(model );
    }
}
  
```

In this case, we just use the default experiment data table, but override some handling of data values for presentation on the table in the `GTableModelProxy` class, which acts as an intermediary between the `ExpDataModel` and the `TableModel`.

### Charts

- `DefaultExperimentBarGraph`
- `DefaultExperimentGraph`
  - This class implements a graphs of X in function Y, where X are all channel and Y is the time
- `DefaultExperimentTimeSeriesGraph`
  - This class implements a series of graphs of X in function Y, where X is the output of a channel and Y is the time
- `DefaultXYExperimentGraph`
  - This class implements a graph of X in function Y, where X and Y are distinct channels

```

public RollPaperXYGraphDisplay () {
    super ();
    setChannelDisplayX ( I );
    setChannelDisplayY ( O );
}
  
```

### Sensor

The Sensor is an additional graphical representation of the results that shows the animation while the experiment is running.

The implementation is typically a Swing component that implements ExpDataDisplay and ExpDataModelListener interfaces.

## Step 4 - Configuration files

### ReCFaceConfig.xml

This file, which should be placed in \$experiment\_name/etc. E.g.,

\$REC\_HOME/experiments/rollpaper/etc/

contains general information related to visual configuration of the experiment, allowing one to configure, for instance, if there is a video associated to the experiment, used to show the execution of the experiment in real time. Watch the configuration file used in the rollpaper experiment:

```
<apparatus location= "ROLL_DOWN_TOILET_PAPER_VI.O"
customizerClassLocationBundleKey= "rollpaper$rec.exp.customizer"
toolTipBundleKey= "rollpaper$rec.exp.tip" iconLocationBundleKey= "rollpaper$rec.exp.icon"
descriptionStringBundleKey= "rollpaper$rec.exp.description"
displayStringBundleKey= "rollpaper$rec.exp.title"
desktopLocationBundleKey= "rollpaper$rec.exp.background" order= "I"
headerDisplayClassLocationBundleKey= ""
displayFactoryClassLocationBundleKey= ""
dataModelClassLocationBundleKey= "" visible= "true" selected= "true"
enabled= "false" connected= "false" >
  <localizationBundle name= "rollpaper"
    location= "recresource:///pt/utl/ist/elab/client/rollpaper/resources/messages" />
    <display classLocationBundleKey= "rollpaper$rec.exp.display.loc.datatable"
      toolTipBundleKey= "rollpaper$rec.exp.display.tip.datatable"
      iconLocationBundleKey= "ReCBaseUI$rec.bui.icon.table"
      displayStringBundleKey= "rollpaper$rec.exp.display.title.I"
      offlineCapable= "false" order= "I" visible= "true" selected= "true"
      enabled= "false" connected= "false" />
  </apparatus>
```

The convention used above to identify the nodes is nameOfLocalizationBundle \$NameOfKeyInPropertiesFile. In the rollpaper experiment, we use as the name of the localizationBundle "rollpaper", to make reference to a key in the file properties, we must use rollpaper \$NameOfKeyInPropertiesFile.

This is closely related to the "name" attribute in the localizationBundle element above, which then states the classpath location of the properties file (without the extension, as it must be composed for different languages, e.g. messages\_en.properties, messages\_pt.properties).

### messages.properties

This file, which should be placed in \$experiment\_name/src/java/client/<full\_package\_to\_experiment>. e.g.,

\$REC\_HOME/experiments/rollpaper/src/java/client/pt/utl/ist/elab/client/rollpaper/resources

is the resource bundle file of the experiment, containing the properties necessary for the internationalization of an experiment (usually, there are two messages files here: messages.properties, with the portuguese translation of the experiment and messages\_en.properties, with the english translation of the experiment)

## Step 5 - Logging

The log initialization for an experience and done as follows:

```
private static Logger LOGGER ;
static {
  final Logger LOGGER = LogManager . getLogManager (). getLogger (EXPERIMENT_NAME );
```

```
if (LOGGER == null){
    LogManager.getLogger().addLogger (Logger.getLogger (EXPERIMENT_NAME));
}
}
```

The log an exception is done as follows:

```
//... more code before this ...
try {
    //...more code here
} catch ( final NullPointerException npe ) {
    LoggerUtil.logThrowable ( "Couldn't get Channel Count from Acquisition Header!", npe, LogManager
        .getLogger ().getLogger (EXPERIMENT_NAME));
}
```

To check if a message of the given level would actually be logged by this logger.

```
//... more code before this ...
if (LOGGER.isLoggable (Level.FINEST)) {
    LOGGER.log (Level.FINEST, "This message is " + " append " + "strings");
}
```

## Step 6 - Internationalization

To adapt the experiments user interface to various languages all the UI resources must be placed in a resource bundle file (usually, there are two messages files here: messages.properties, with the portuguese translation of the experiment and messages\_en.properties, with the english translation of the experiment). E.g.,

```
rec.exp.icon.rollpaper=recresource\:///pt/utl/ist/
elab/client/rollpaper/resources/pend2m_iconified.PNG
rec.exp.tip.rollpaper=Experiência sobre Queda de Rolo de Papel Higiênico
rec.exp.customizer.rollpaper=pt.utl.ist.elab.client.rollpaper.RollPaperCustomizer
rec.exp.title.rollpaper=Deixar cair rolo de papel
rec.exp.description.rollpaper=<html><body>Esta é uma descrição
da experiência Roll down Toilet Paper. <br>Aproveite.</
body></html> rec.exp.background.rollpaper=recresource
\:///pt/utl/ist/elab/client/rollpaper/resources/mec8.gif
rec.exp.display.rollpaper.loc.datatable=pt.utl.ist.elab.client.rollpaper.displays.DataTable
rec.exp.display.rollpaper.tip.datatable=Dados RollPaper
rec.exp.display.rollpaper.title.1=Tabela de Dados
```

**Resource Types:**

- String
- ImageIcon (Prefixed with recresource\; )
- Object
- URL (Prefixed with the url protocol, E.g., http\; )
- Support to HTML 3.2
- Not support to JavaScript

These files, should be placed in \$experiment\_name/src/java/client/  
<full\_package\_to\_experiment>.resources E.g.,

```
$REC_HOME/experiments/myexperiment/src/java/client/pt/utl/ist/elab/client/
myexperiment/resources/messages.properties
```

The resources are retrieved with the ReCResourceBundle utility. E.g.,

```
ReCResourceBundle . findString ( "rollpaper$rec.exp.lbl.samples" );
```

Please check the find\* methods to choose the one that fits the resource type to get.

## Step 7 - Deployment

TBD

## Step 8 - Ant Targets

To run a experiment, in \$REC\_HOME using the following command:

```
ant -f experimentsbuild.xml run.experiment.client -  
Dexperiment.name=<experiment_name>
```

To debug a experiment, in \$REC\_HOME using the following command:

```
ant -f experimentsbuild.xml debug.experiment.client -  
Dexperiment.name=<experiment_name>
```

For more information visit \$REC\_HOME/experimentsbuild.xml.

## Step 9 - Hardware Communication