EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Control System for the CHORUS Microscope Lab

Daniele Raffo
Universite de Marne la Vallee, France

August 16, 2001

Contents

Chapter 1

Introduction

This is a concise instruction manual for the set of program I wrote (in Java 1.2.2) for the Microscope Laboratory of the CHORUS project, EP Division.
My set of program has been issued from the old Monitor program (see “A Monitor program for the Microscope lab” by Christian Schmitt, September 17, 1999), that contained several functionalities for Microscope control, data monitoring, Slow Control, and Dispatcher messages checking.
To keep the entire structure simple, and for increased stability and performances, I have redesigned the whole project as a set of indipendent programs, each one devoted to a different functionality. Hence this set of programs is composed by three modules: the Control Panel, the Dispatcher Console and the Alarm Checker. Each one of them is a standalone program and may be used indipendently of the others (although they have been designed to work in symbiosis).

Chapter 2

The program set

2.1 Control Panel

The Control Panel program is the module used for controlling a Microscope: he sends messages to it through the Dispatcher, and handles messages in feedback. New features (low-level configuration) have been added. The Acquisition Setup panel is now (with the exception of the Plate selection panel) wholly built in a dinamical way, from the data contained into an ASCII file loaded at the start of the program. Furthermore, as the Control Panel is supposed to be used also by untrained users, the program contains a system for specifying constraints (into another ASCII file) for the Plate selection panel. This will avoid entering incoherent values that may result in wrong data from the Microscope. A system for storing commands that have to be sent automatically along with a `CONST SET type` message, has been added too. An online help makes using the program much easier.

2.2 Dispatcher Console

The Dispatcher Console may be utilised to check the messages that comes to and from the Dispatcher. It is a simple program whose only purpose is to show messages having a specific tag (through the subscription mechanism), and hence acts solely as a receiver. Useful for debugging.

2.3 Alarm Checker

The purpose of the Alarm Checker module is to perform a constant check over different parts of the Microscope Lab system, and alert the user if something goes wrong. User is alerted by message dialogs and/or via email (making use of the JavaMail API).

Chapter 3

Usage

You should run the Alarm Checker first, to monitor the behaviour of the other modules. Then you run a copy of the Control Panel for each Microscope you want to control. The Dispatcher Console program may be started and killed at any time.

For the first time, I suggest running the programs with no configuration file (just specify an inexistent one as argument), so they will boot with default values. Then, configure the programs as you like[1], save, and ther rerun them by specifying as argument the configuration file you just wrote.

For the purpose of testing, I wrote also Send Dispatcher, a tiny command-line program (no GUI) that may be used to send messages to the Dispatcher. The synopsis is:
`java ch/cern/chorus/emulsion/program/SendDispatcher` *`host port tag message`*
At the time I am writing, the Dispatcher runs on host sunch02.cern.ch on port 5553, so you should try using these values.

If you aren't able to run these programs and get a Java Exception `java.lang.NoClassDefFoundError` instead, check your `CLASSPATH` environment variable. It should look like this one:
`.:/afs/cern.ch/user/d/`*`yourlogin`*`/CVSWork/Java:`
`/afs/cern.ch/chorus/micprogs/Java/javamail-1.1/mail.jar:`
`/afs/cern.ch/chorus/micprogs/Java/jaf/activation.jar`

[1] For the Alarm Checker, remember to define the actions to take in case of alert, too.

Chapter 4

Contact

For any question, comment or bug report, feel free to contact me at one of the following addresses:

Daniele.Raffo@cern.ch

draffo@etudiant.univ-mlv.fr

danraffo@yahoo.it

Furthermore, I will be at CERN, office 1-1-041, until August 31st, 2001.

Part I

Control Panel

Chapter 5

The program

When running the program for the first time, it is important to change the configuration by the relevant menu option, and correct the paths from `/afs/cern.ch/user/d/draffo/CVSWork/Java/ch/cern/chorus/emulsion/program/cfg/` to as necessary.

The Control Panel program must be run with the following syntax:
`java ch/cern/chorus/emulsion/program/ControlPanelProgram` *`tag cfgfile`*
where *tag* is the tag related to the Control Panel, and *cfgfile* is a Java Properties file containing the configuration that has to be used. If *cfgfile* does not exists or is unreadable, the program will start with default values.
The *tag* will then identify in an unique way this processus: to control multiple microscopes, just run multiple instances of the Control Panel program, each one with a different value of *tag*. All messages sent to the Dispatcher from the same instance of the program will then all have the same tag.
The value of *tag* should be `PANEL`*`xx`*, where *xx* = `01`, `02`, `03` ...
For its communications with the Microscope, the Control Panel will then consider the messages coming with tag `MICOS`*`xx`*.

5.1 Configuration

You may change the configuration by selecting Change configuration from the menu.
On the panel, roll the mouse over a field's name to get a brief explication of the

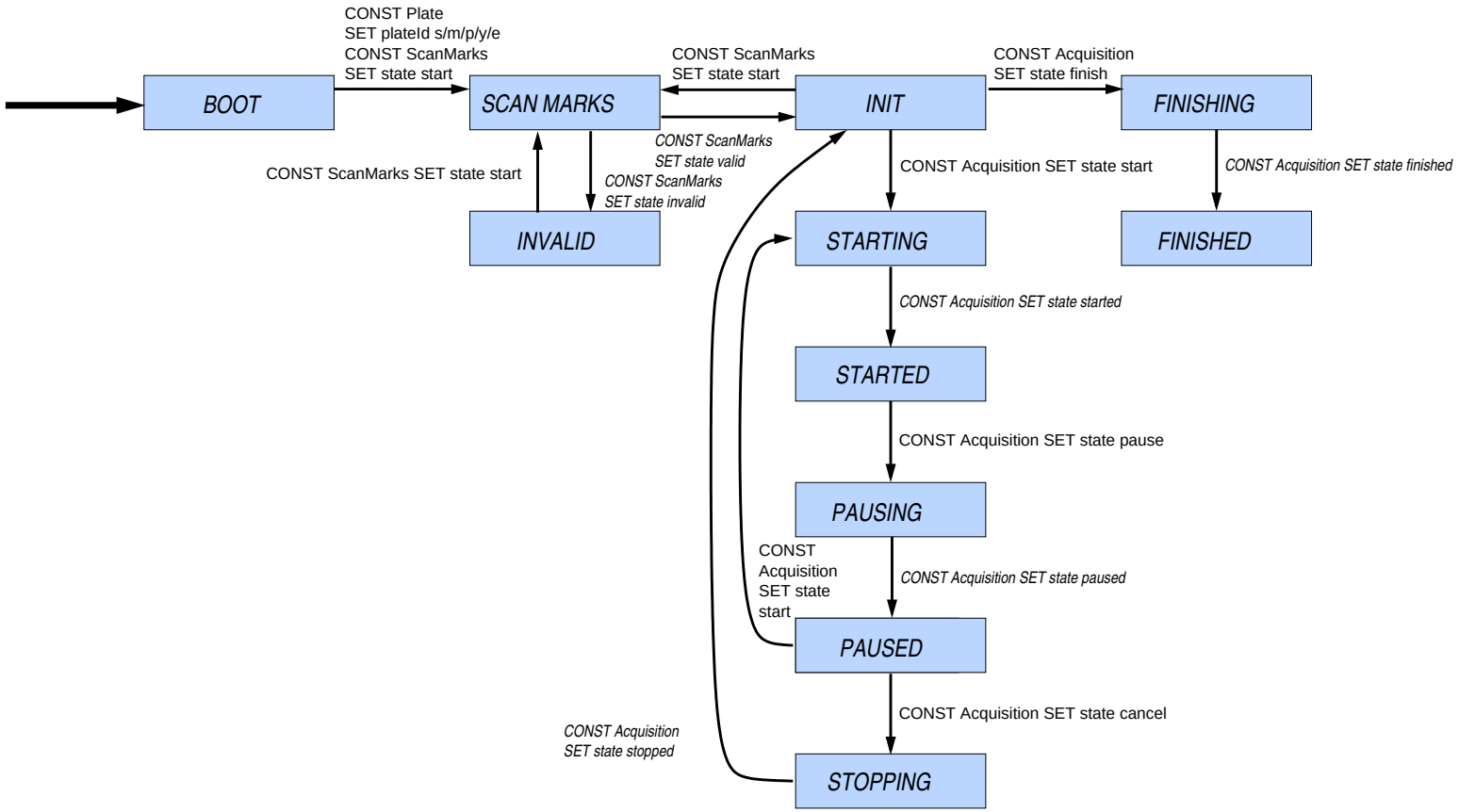
field. Once you've modified the fields you want, you may write this configuration into a file by specifying its name into the lower field, and then clicking on the Save in... button near to it. The changes in the configuration will not take effect during this session. You may use the new configuration by running again the program, specifying the name of the configuration file as the *cfgfile* argument.

5.2 Control Commands

Here are the buttons to control the Microscope: Scan marks, Start/Continue, Pause, Stop and Finish. The four latter will send the corresponding messages `CONST Acquisition SET state` *`state`*.
Over the control buttons, a text informs you constantly about the current status. It may be useful to remark that when the Control Panel is awaiting messages from the Microscope, the text ends always with three dots and all buttons are disabled.

In case something goes bad (wrong messages from Dispatcher, etc.) and you need to resume the situation, you may utilise the option Enable all buttons in Control Commands under the Panic menu. **Use this feature cautiously:** it may bring the Microscope into an unstable state.
The Panic menu allows the user to issue recovery commands in emergency situations; the commands listed there should be used only when really necessary, and not during normal behaviour.

The next page shows the state diagram of the Control Panel / Microscope system. Messages from Control Panel to Microscope (tag `PANEL`*`xx`*) are in plain text, and messages from Microscope to Control Panel (tag `MICOS`*`xx`*) are in italic.


CONST Plate
SET plateId s/m/p/y/e
CONST ScanMarks
SET state start
CONST ScanMarks
SET state start
CONST Acquisition
SET state finish
BOOT
SCAN MARKS
INIT
FINISHING
CONST ScanMarks
SET state valid
CONST ScanMarks SET state start
CONST ScanMarks
SET state invalid
CONST Acquisition SET state start
CONST Acquisition SET state finished
INVALID
STARTING
FINISHED
CONST Acquisition SET state started
STARTED
CONST Acquisition SET state pause
PAUSING
CONST
Acquisition
SET state
start
CONST Acquisition SET state paused
PAUSED
CONST Acquisition SET state cancel
CONST Acquisition
SET state stopped
STOPPING

5.3 Joystick panel

A joystick panel will pop up when the Microscope sends a message `CONST Joysticks SET state started`.[1]
The arrows allow you to move the Microscope in the six directions, plus four diagonal. Press the arrow button to start the movement, and release it to stop. With the slider on the top of the panel you can change the movement speed.
You are not supposed to close the joystick panel on your own; it is removed automatically when a message `CONST Joysticks SET state finished` is received from the Microscope.[2] A message dialog will warn you about that.

5.4 Console

You may send any message to the Dispatcher, by entering it into the text field, and clicking on Send message or just by hitting Return.
Remember that the tag of the message will be `PANEL`*xx*, the tag identifying the Control Panel you are running; you do not have to specify it again.

5.5 Message History

It shows all messages sent to the Dispatcher during the session. The history window is cleared after that a maximum of `max-history-size` messages are attained, or when you select the Clear history option from the menu. (Alternatively, you may select this option from a pop-up menu too, by clicking the right mouse button on the history window).

5.6 Acquisition Setup

It contains the Plate panel for the selection of the plate to scan, and a number of other panels built dinamically at startup.

[1] If necessary, the joystick panel may also be brought visible by issuing the command Show joystick from the Panic menu, although this is not the normal way.

[2] However, you may always force the closing of the joystick panel via the standard X Window "close" button, on the side of the title bar.

5.6.1 Plate

From here, you can select the values for Stack, Module, Plate, Year and Period, and then send them to the Dispatcher by clicking on Set.
The value you have selected for Stack affects the possible choices you'll have for the Module; similarly, the Year is determined by your choice for the Period, and the constraints between Year and Period are determined by the value you choose for Plate.
The program activates these constraints by reading them from `constraints-file`. See chapter 7, "Low-level configuration: Specifying constraints" to know how to write a such file.

5.6.2 Sections

These panels (each one of them represents a different Section) are built dinamically following the directives of the `gui-settings-file`. See chapter 6, "Low-level configuration: Sections in Acquisition Setup" to know how to write such a file.

Setting an option

Each Section contains a JComboBox for the selection of an option, and two buttons: Set and Setup.
Select an option; on the textfield below will appear some help about the option you are currently selecting. Clicking on Set will now make the program sending the message `CONST` *`sectionname`* `SET type` *`optionselected`* to Dispatcher. Besides, this will enable the Setup button.

Option Setup

Clicking on Setup will bring a Setup window from where you can change the value of different parameters, related to the Section and option selected. (On the header of the window will actually appear: Setup for *Section : option*).
Rolling the mouse over a parameter will make the bottom textfield display some help about it. Once you've changed one or more values, click on the button Set of this window to send a `CONST` *`sectionname`* `SET` *`parametername parametervalue`* message for each one of the parameters. Clicking Close will close this window without sending any message. When you will reopen this

window again, you'll find the last values you entered.
The Default button will reset all parameter's values to their default, as specified in the `gui-settings-file` (these are the values that appeared the first time you opened that window in this program's session).

Sending your own parameters values at start

You might want to memorize a whole set of values, to be sent automatically to Dispatcher when you Set an option.
To do that, in the Setup window for the Section and option you've chosen, enter the desired values for each parameter. Then, tick the Send always as start checkbox and click on the Save start button. The button will become orange to show that you've memorized your own defaults for this option. Click on Close.
If you reselect this option in the right Section of the Acquisition Setup, you will notice that the option's name in the JComboBox is paint in orange colour. Now, clicking on the Set button will send the message `CONST` *`sectionname`* `SET type` *`optionselected`* , followed automatically by the whole lot of messages `CONST` *`sectionname`* `SET` *`parametername parametervalue`* for each one of the parameters.

When you don't want anymore to use your own defaults whenever you Set this option, you have only to go to the Setup window, untick Send always as start and click on Save start. The button will become light orange to show that your own defaults you had previously memorized have been disabled (but they haven't been destroyed, and may be enabled again at any time by simply re-ticking Send always as start and clicking on Save start).
Now, when you reselect this option in Acquisition Setup, it will be painted in light orange, and clicking on the Set button will make the same behaviour as before - only a single message will be sent.

Once you've clicked on the Save start button, the Start default button becomes clickable, whether your own defaults are enabled or not. The latter button acts exactly as the Default button formerly described, but it resets all parameter's values to your own defaults instead.

These own defaults are written on disk (on the file specified in the configuration property `saas-defaults-file`), and hence will be reloaded at the program's next session.

Chapter 6

Low-level configuration: Sections in Acquisition Setup

If you are only an user of the Control Panel, and do not need to configure it, you may skip this chapter.

The group of Sections in the Acquisition Panel is described into an ASCII file, whose name is stored in the configuration property `gui-settings-file`, and having the following structure:

```
descriptor of section #1, option #1
     descriptor of parameter #1
     descriptor of parameter #2
     descriptor of parameter #3
     descriptor of parameter #4
     ...
descriptor of section #1, option #2
     descriptor of parameter #1
     descriptor of parameter #2
     descriptor of parameter #3
     ...
descriptor of section #1, option #3
     descriptor of parameter #1
     descriptor of parameter #2
     ...
...
descriptor of section #2, option #1
     descriptor of parameter #1
     descriptor of parameter #2
     ...
descriptor of section #2, option #2
     descriptor of parameter #1
     descriptor of parameter #2
     descriptor of parameter #3
     ...
...
```

Here is an example of a such file. When feeding the program with this file, it will result in an unique section in the Acquisition Setup; it will be named Criterion and have two possible options: CHORUS_CS and NetScan. The former will have as parameters input a boolean, two textfields and two lists, while the latter will have a boolean and a list.

```
// @section: Criterion
// Selects predictions on changeable sheet (CS) for vertex location
// scanback.Normal scanning procedure should be TwoSidesIndependent.
[ CHORUS_CS ]

// @type: boolean
// Do not select predictions that were already scanned
skipScanned = yes

// @type: text
// Defines the minimum and maximum value of the error on the prediction.
// If the squared error is smaller or larger then the prediction is
// not scanned;
posErrorRange   = 0 400

// @type: text
// Gives the range on the plate that can be scanned, usually; Two pairs of
// two numbers; The first pair gives the left-bottom coordinates, the 2nd
// pair the right-upper coordinates;
positionRange   = 16000 -714000 358000 -20000

// @type: list DOWNSTREAM UPSTREAM
// The side of the emulsion plate that is facing upwards on the stage. For
// Changeable and Special Sheet this is normally the DOWNSTREAM surface.
// Bulk plates are reversed because only the UPSTREAM 100 micron is scanned.
topSurface      = DOWNSTREAM

// @type: list DOWNSTREAM UPSTREAM BOTH
// The side of the emulsion plate that needs to be scanned;
sideToScan      = UPSTREAM

// @section: Criterion
// Selects predictions for
// NetScan; Normal scanning procedure should be OneSide
// using a GrainMap tracker
[ NetScan ]

// @type: boolean
// Do not select predictions that were already scanned
skipScanned     = Yes

// @type: list DOWNSTREAM UPSTREAM
// The side of the emulsion plate that is facing upwards on the stage. For
// Changeable and Special Sheet this is normally the DOWNSTREAM surface.
// Bulk plates are reversed because only the UPSTREAM 100 micron is scanned.
topSurface      = DOWNSTREAM
```

6.1 Descriptor of Section and option

This *descriptor* is a block of lines, structured as follows:

- One line beginning by `//␣@section:␣` , followed by the name of this Section;
- One or more lines beginning with `//␣` , containing a comment for the option of this section. Don't worry about line breaking; the program will reformat the input to make it fit into the help window;
- One line composed of an option for this Section, prepended by `[␣` and followed by `␣]` (that is, surrounded by square brackets and a space).

6.2 Descriptor of parameter

Also this *descriptor* is a block of lines, structured as follows:

- One line beginning by `//␣@type:␣`, followed by the type of the input that may be `boolean`, `text` or `list` (which will be respectively rendered in the Java Swing GUI by a JCheckBox, a JTextField or a JComboBox). If the type is `list`, you need to prepend also the names of the elements of the list, separed by one space;
- One or more lines of comment for this parameter, as described above;
- One line composed by this parameter's name, followed by at least one space, an `=` sign, another space, and the default value for this parameter.
  If the type of input was `list`, the default value should be an element of the list. If the type of input was `boolean`, the default value must be a boolean (the *true* value is matched by any of the following keywords: `TRUE`, `YES`, `Y`, `ON` and `1`, case insensitive).

There must be always one and only one blank line between a *descriptor* block and another. Also, pay attention to spaces inside lines, they are important.

Chapter 7

Low-level configuration: Specifying constraints

If you are only an user of the Control Panel, and do not need to configure it, you may skip this chapter.

The constraints for the Plate selection panel are described into an ASCII file, whose name is stored in the configuration property `constraints-file`, and structured as follows.
Some examples will explain the instructions that may be used in that file.

7.1 Stack / Module constraints

To specify that Stack 21 contains only the Module 3, and Stack 22 contains only the Modules 3 and 4, use these instructions in the file:

```
STACK 21 ALLOWS MODULE 3
STACK 22 ALLOWS MODULES 3 4
```

with the Module numbers separated by at least one space.

7.2 Plate set

To indicate that the Plate set, goes from plate -2 to plate 36, use the following instruction:

```
PLATE RANGE -2 36
```

If the Plate set does not contains a continue range, eg. is composed by plate -2, -1, 0, 5 and 10, use the following:

```
PLATES -2 -1 0 5 10
```

Note that plates may be identified by a name as well as a number.

```
PLATES CS SS 0 5 10
```

The order in which are specified the range limits is not important. Hence the following three instructions are equivalent:

```
PLATE RANGE -2 5
PLATE RANGE 5 -2
PLATES -2 -1 0 1 2 3 4 5
```

7.3 Period / Year constraints, depending of the Plate selected

Let's say that, for the Plates -2 and -1, the Periods 21 and 22 belongs to the Year 96, and the Period 23 belongs to the Year 97.
You have then to declare a Period / Year Specification, and you must attribute a number to it (here we suppose 1):

```
SPEC 1
PERIOD 21 22 23
YEAR   96 96 97
```

Each Specification is made up of three consecutive lines:

- The first line must begin with the keyword SPEC, followed by a positive, non-zero number that identifies the Specification;
- The second line begins with the keyword PERIOD, followed by the Periods allowed;
- The third line begins with the keyword YEAR, followed by the Year that applies to the corresponding Period of the previous line.

Note that you don't have to put two extra spaces after the keyword YEAR; this is done only for readability's sake. However, it is necessary that the PERIOD and the YEAR lines have the same number of values.

Once you have done that, you're ready to attribute this Specification (that, I remind, you have chosen be identified with 1) to Plates -2 and -1.
This is made simply via the SET SPEC instruction:

```
SET SPEC 1 PLATES -2 -1
```

with each Plate number separated from the others by a space.

By these instructions, when Plate -2 or -1 is selected, and the user selects Period 21 or 22, the Year will be then automatically set to 96. When the user selects Period 23, the Year will be set to 97. No different choices will be allowed for the Period.

7.3.1 The default Period / Year Specification

You may create as many Specifications as you want, as long as you name every one of them with a different number.
Obviously, every Plate has to be attached to a Specification. To make things easier, I designed this model where it is required that you create at least the default Specification. This is a Specification that is identified by the keyword DEFAULT:

```
SPEC DEFAULT
```

followed, as usual, by the PERIOD line and the YEAR line.
You may also attribute the identifier 0 to such a Specification.
This default Specification will then be automatically attached to every Plate for which you didn't set explicitely a Specification.

7.3.2 To allow the selection of the Year

To allow the user to select an Year, use the DONTCARE keyword instead of an Year number, in correspondence of a Period number:

```
SPEC 2
PERIOD 21 22 23 0
YEAR   96 96 97 DONTCARE
```

By doing this, you allow the user to choose an Year when the corresponding Period is selected (in this case, Period 0). The choices for the Year will then include all the Years you mentioned in the YEAR line, in all Specifications you created.

7.4 Example

Here follows an example of a complete constraints file.

```
# Constraints configuration file

STACK 21 ALLOWS MODULE 3
STACK 22 ALLOWS MODULES 3 4
STACK 23 ALLOWS MODULES 4 5 6 7
STACK 25 ALLOWS MODULES 4 5 6 8 11 12


PLATE RANGE -2 36


SPEC 1
PERIOD 21 22 23
YEAR   96 96 97

SPEC 2
PERIOD 21 22 23 0
YEAR   96 96 97 DONTCARE

SPEC DEFAULT
PERIOD 21 22 23 0
YEAR   96 96 97 0


SET SPEC 1 PLATES -2 -1
SET SPEC 2 PLATE 0
```

7.5 Miscellaneous

To enter a comment, begin a single line with the character #, `//` or `/*`:

```
# This is a comment
```

With the exception of the PLATE instruction, that must always be declared before the SET SPECs, all others instructions may be in any order on the file. Even the SET SPEC command may be used *before* the declaration of the specification body (SPEC).
Furthermore, instructions of the same type (eg. STACK) does not need to be declared together; although, for clarity, I suggest to do so.

Instructions may be separated by blank lines. The exception to this rule is the SPEC instruction that needs to be immediately followed by the PERIOD and the YEAR lines.

All commands may be uppercase as well as lowercase.

You may indifferently use the keywords PLATE or PLATES; the same goes for the keywords MODULE and MODULES.

Part II

Dispatcher Console

Chapter 8

The program

When running the program for the first time, it is important to change the configuration by the relevant menu option, and correct the paths from `/afs/cern.ch/user/d/draffo/CVSWork/Java/ch/cern/chorus/emulsion/program/cfg` to as necessary.

The Dispatcher Console program must be run with the following syntax:
`java ch/cern/chorus/emulsion/dispatcher/DispatcherWindow` *`cfgfile`*
where *cfgfile* is a Java Properties file containing the configuration that has to be used. If *cfgfile* does not exists or is unreadable, the program will start with default values.

The Dispatcher Console will then intercept and shown the messages whose tags are listed in the configuration property `subscribe-tags`. You may change this value by the apposite menu option, as described later.
If you want, you may also enter the tags, to which you want to subscribe, on command line. Just use the following syntax[1]:
`java DispatcherWindow` *`cfgfile`* `tags` *`tag1 tag2 ...`* These tags will then override the old value of `subscribe-tags`.

[1] Example: `java DispatcherWindow mycfg tags MICOS01 TEST00`

8.1 Main window

The main window shows the messages that are sent to the Dispatcher, with their datestamp and tag. The subscription tags are shown, as a remainder, in a line under the main window.

When the number of messages in the window reaches the value of the configuration property `max-buffer-size`, the oldest three messages are deleted. You may clear the entire buffer at any time by selecting the Clear window option from the menu. The same option may be selected from the main window by its popup menu, obtained by clicking the right mouse button.

8.2 Configuration

You may change the configuration by selecting Change configuration from the menu.
On the panel, roll the mouse over a field's name to get a brief explication of the field. The property `subscribe-tags` lists the tags, separed one from each other by a space, to which a subscription has been made.
Once you've modified the fields you want, you may write this configuration into a file by specifying its name into the lower field, and then clicking on the Save in... button near to it. The changes in the configuration will not take effect during this session. You may use the new configuration by running again the program, specifying the name of the configuration file as the *cfgfile* argument.

Part III

Alarm Checker

Chapter 9

The program

When running the program for the first time, it is important to change the configuration by the relevant menu option, and correct the paths from `/afs/cern.ch/user/d/draffo/CVSWork/Java/ch/cern/chorus/emulsion/program/cfg` to as necessary.

The Alarm Checker program must be run with the following syntax:
`java ch/cern/chorus/emulsion/alarm/AlarmChecker` *`cfgfile`*
where *cfgfile* is a Java Properties file containing the configuration that has to be used. If *cfgfile* does not exists or is unreadable, the program will start with default values. Please note that the program will start with default values also if *cfgfile* is valid but `alarms-action-config-file` has not been set yet (see section 9.2).

9.1 Main panel

The program shows the list of events that may trigger an alarm, as well as the checking status (checking / checking disabled / interval time).
On the bottom a window shows the alarms that have been triggered, with date and time. This window can be cleaned by selecting Clear window from the menu, or from a popup menu (obtained by clicking with the right mouse button on the window). The window will anyway be cleaned automatically after that the number of events reaches the value specified in the `triggered-alarms-bufsize` configuration property.

The Alarm Checker will check for the following events:

Slow Control. Checks for temperature and pressure values in the Microscope Lab, and triggers an alarm if any of these values are out of range. *Not implemented yet, due to problems with hardware connections.*

No messages from Dispatcher. Triggers an alarm when there is no message from the Dispatcher since a given amount of time. This amount of time is configurable from the panel Define actions to take. The configuration property `dispatcher-alltags` contains the tags that are listened for; the "no messages" time is computed only with respect to these tags. If there are no messages for the given time from a certainly tag, but this tag is not listed in `dispatcher-alltags`, no alarm will be triggered.

Control Panel and Slow Control clients. Triggers an alarm whenever a Control Panel or Slow Control program is started or ends, anywhere in the network. It will give informations about who started the client, and from which host.[1] *Note: the detection of Slow Control clients is not implemented yet.*

Message of "Plate finished". Triggers an alarm whenever is intercepted a message `CONST Plate SET finished`, indicating that the current plate needs to be changed. For being intercepted, this message must come with a tag that is listed in `dispatcher-alltags`, so you must take care to set properly this configuration property.

Custom alert messages. You may trigger your own custom alarms, simply by sending a message to the Dispatcher having one of these tags: `ALARM01 ALARM02 ALARM03`. The alert will then contain the message tag as source, and the message text as information.[2]

9.2 Defining actions to take when an alarm is triggered

It is possible to specify what to do when an alarm is triggered, by the menu option Define actions to take. This will bring to a panel showing 11 lines; every line

[1] This is done by listening the tags `Born` and `Died`, generated internally by the Dispatcher. Because of that, the AlarmChecker won't be able to detect any client that was started *before* the AlarmChecker itself.

[2] Example: `ALARM01 There is an alligator in the Microscope Lab!`

belongs to a different email recipient, specified in the Email field at the end of the line. You may select the events you want to be warned of, by ticking the checkbox that belong to that event. An alert message will then be sent to the specified email address when any of the ticked events should occur. (You may also receive alerts via SMS on CERN mobile phones, but this is purely a CERN facilities feature.)
The meanings of the event labels are:

slow ctrl Slow Control values out of range

no msgs No messages from Dispatcher

CP birth Launch of a Control Panel client

CP death End of a Control Panel client

SC birth Launch of a Slow Control client

SC death End of a Slow Control client

plate Message of “Plate finished”

alarm0X Custom alert

For the alarm0X custom alert, there are three checkboxes: they concern the messages with tag, respectively, `ALARM01`, `ALARM02` and `ALARM03`. Therefore you can have different behaviour and email recipients for each one of these tags.
The eleventh line does not contain a field for email address and is labelled Onscreen. This line concerns warnings onscreen; the program, instead of sending an email, will pop up an alert message dialog.
A field named Messages delay allows to specify the time (in seconds) between two messages from Dispatcher, with respect to the no msgs alert.

When an event happens, it is reported in the Triggered alarms window on the main panel. In addiction of that, depending of the choices that have been made on the Define actions to take panel, a message dialog will pop up and/or an email will be sent to the concerned person(s).
If any error should be encountered while sending the emails, a message window will inform of the problem.

Some notes regarding message dialogs.
When an alert message dialog pops up, events of that particular type won't be intercepted anymore until you acknowledge it by pressing OK on the requester. However, events of different type will still be intercepted.
When more than one message dialog appears, they must be closed (by clicking on OK) in inversed sequence, from the last to the first.

After making your choices, the Apply button allows to apply them immediately, without exiting from the panel. The Use button will apply the choices for the current session only. Clicking on the Save and use button will apply the choices for the current session, and save them into the file specified in the configuration property `alarms-action-config-file`. Otherwise, the Cancel button will cancel any changes you have made.

9.3 Configuration

You may change the configuration by selecting Change configuration from the menu. On the panel, roll the mouse over a field's name to get a brief explication of the field. Once you've modified the fields you want, you may write this configuration into a file by specifying its name into the lower field, and then clicking on the Save in... button near to it. The changes in the configuration will not take effect during this session. You may use the new configuration by running again the program, specifying the name of the configuration file as the *cfgfile* argument.

Part IV

Appendix

Appendix A

Classlist

Here is the complete list of every new class I wrote, with some informations about it. The classes are regrouped by package; if the package itself has been created ex novo by me, there is some informations about it too. Classes have also comments and javadocs in them. Several of the old existing classes have also been modified and extended in a minor way.

This part of documentation is intended for developers only. If you are not willing to modify or examine the sources, you don't need to read this appendix. (It would look pretty cryptic to you, anyway...)

A.1 ch.cern.chorus.emulsion.alarm

The classes of the Alarm Checker program.

A.1.1 ActionsElem.java

GUI component. One single line of the ConfigActionsPanel. Contains the JCheckBox's and, depending of the constructor used, a JTextField for the email address or simply a JLabel "Onscreen".

A.1.2 AlarmChecker.java

The Alarm Checker main program. Builds the GUI, manages the triggering of the different alerts.

A.1.3 CheckNoMessages.java

Run from the Alarm Checker main program, checks every 500 msecs if the time of the last message from Dispatcher is older than the interval specified, and triggers an alarm if it is the case. Uses multithreading.

A.1.4 ConfigActionsPanel.java

GUI component. The panel for configuring the actions to take when an alarm is triggered. Contains eleven ActionsElem units.

A.1.5 EmailMessage.java

Uses the JavaMail API. Creates and sends an email message, following the directives of the AlarmChecker. This is done in background by another thread, to avoid locking the main program flow.

A.1.6 InfoAlarmsPanel.java

GUI component. The panel into the Alarm Checker main window showing the alarms handled.

A.2 ch.cern.chorus.emulsion.controlpanel.panels

A.2.1 CommandPanel.java

GUI component. The part of the Control Panel that contains the control buttons and the information text about the current status. Manages the transitions between system states.

A.2.2 CustomSettingsPanel.java

GUI component. The panel Acquisition Setup in the right part of the Control Panel main window. Holds the PlatePanel, and the Section's that will be built dinamically from ASCII file.

A.2.3 HistoryPanel.java

GUI component. Records all messages sent from the Control Panel to the Dispatcher.

A.2.4 PlatePanel.java

GUI component. Builds the Plate panel in the Acquisition Setup, following the directives computed by the ConstraintsCompiler. It also manages the dynamic modifications of the items in the JComboBox's, during the user's selection.

A.2.5 SendRawPanel.java

GUI component. The panel allowing to send a custom message from the Control Panel to the Dispatcher.

A.2.6 TitlePanel.java

GUI component. The title panel in the header of the Control Panel main window. Shows tag, user, and time start of this Control Panel session.

A.3 ch.cern.chorus.emulsion.dispatcher

A.3.1 DispatcherWindow.java

The Dispatcher Console main program. Builds the GUI and listens to the Dispatcher for the specified tags.

A.4 ch.cern.chorus.emulsion.program

Contains the main Control Panel program class, and some other important classes.

A.4.1 ConfigurationPanel.java

A GUI panel allowing to change the configuration, stored as a Java Properties, in a graphical way. Its constructor takes as argument a Properties file; then it builds a GUI with a JTextField for each property value found. For its flexibility, it is used

by the Control Panel, the Dispatcher Console and the Alarm Checker. Provides also a help feature via JToolTip.

A.4.2 ControlPanelProgram.java

The Control Panel main program.

A.4.3 JoystickWindow.java

GUI component for the joystick panel.

A.4.4 SendDispatcher.java

A little standalone shell program, that can be used to send a message with any tag to the Dispatcher.

A.5 ch.cern.chorus.emulsion.program.constraints

Package for the Plate constraints system.

A.5.1 ConstraintsCompiler.java

Parser for a Plate constraints ASCII file.

A.5.2 Specification.java

A period - year specification in the Plate constraints structure.

A.5.3 UnrecognizedCommandException.java

Exception thrown by the CostraintsCompiler if there is an incomprehensible line in the constraints file under analysis.

A.6 ch.cern.chorus.emulsion.program.guibuilder

The package for the interpretation of the directives and the dynamic building of the Acquisition Setup panel.

A.6.1 ParameterEntry.java

GUI component. A single element of the SetupPanel; it allows the configuration of a single parameter. The input device may be a JCheckBox, a JTextField or a JComboBox.

A.6.2 Section.java

GUI component. A customized Section in the Acquisition Panel, which will then contain the PlatePanel and some Sections.

A.6.3 SettingsReader.java

Parses the ASCII file containing the directives for the Sections. The complete structure is as follows:
Each Section contains a number of differents types, from which the user can select, via a JComboBox, the preferred one. These types are stored in an Hashtable (as a field of the Section) which contains for each type (key) the corresponding SetupPanel (value).

A.6.4 SetupPanel.java

GUI component. Allows the configuration of the parameters of a determined type in a Section. It is made by one or more ParameterEntry's.

A.7 ch.cern.chorus.emulsion.util

Package providing some useful classes for general purpose.

A.7.1 Bool.java

Parses a String and interprets its boolean value, and viceversa.

A.7.2 JTextInfoArea.java

Contains a JTextArea with scrollbars; it is utilised to provide a help area for the Setup Panel and the Section elements of the Acquisition Panel.

The following directories does not contain any Java class. Instead, they contain some resources needed or useful in order to use the set of programs.

A.8 ch/cern/chorus/emulsion/program/cfg

Program icons and examples of configuration files.

A.9 ch/cern/chorus/emulsion/docs

This documentation.