# MBE Growth Control Program

Enrico Segre, `enrico.segre@weizmann.ac.il`

version 1.0.2, commit 408547, May 5, 2020

# Contents

# Part I

# User manual

# Chapter 1

# Scope and purpose of the program

This is a program for controlling molecular beam epitaxial plants, developed for the Submicron Center at the Weizmann Institute, and therefore geared *in primis* for the present hardware and working practices. As such, this program allows to define the plant configuration, monitors and controls the immediate state of effusion cells and shutters; it is then devised for writing recipes for layered growth processes, and to execute them in automatic or supervised mode.

The program is designed to run also in absence of connected hardware, to allow editing and reviewing of growth sequences.

In which order to read this document: Part I is more intended as an user manual: common everyday use of the program involves designing growth recipes, as described in 3.3; validating and graphically reviewing growth recipes (3.3.4); and monitoring the plant hardware during idle, preparation and growth phases, as described in 3.5. The general configuration of the plant, described in 3.2, needs to be created once from scratch for a new plant, and would seldom be revised when hardware is recalibrated or changed. The installation of the program on a computer and its requirements are explained in 2.

Part II discusses a number of program design issues, and rambles about choices done in the development of this program.

# Chapter 2

# Installation

Windows installers for the compiled program are provided (executables for linux and Mac could also be generated on request). The installers are generally named according to the short git SHA1 signature of the snapshot built (see 8.1).

The default application directory proposed by the installer is `C:\Program Files\GrowthScheduler`.

## 2.1 Executable Build Installer and Runtime Engine

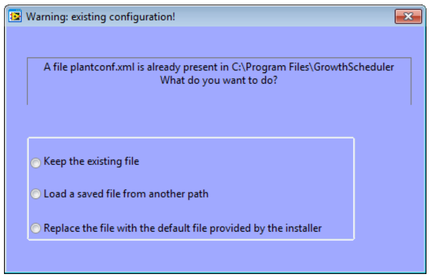
The program is coded in LabVIEW, hence even the compiled package requires the preliminary installation of a LabVIEW runtime engine. As of the core development in 2015, the program has been developed using LabVIEW 2014sp1, so the runtime engine needed is the corresponding one, which can be downloaded from ni.com site or installed from LabVIEW media. Sporadically I provide packaged installers including the engine too (size ~350MB), but otherwise, for minor and continuing updates, I just generate a smaller (~14MB) installer for the compiled code alone.

The Runtime engine needs to be installed only once on the target machine. It can be installed running once the full installer of an older version; the application can be subsequently updated by a later smaller installer. The runtime engine is also present whenever the LabVIEW IDE has been installed.

## 2.2 Updating previous distributions

The same installer can be used to create an istallation *ex novo*, as well as to upgrade the a previous version. According to Windows procedures, downgrading is only possible first uninstalling (Control Panel/Programs and Features) the more recent version and subsequently reinstalling the older[1].

Each installer includes, for definiteness, a given version of the plant configuration file `plantconf.cfg` (3.2), and copies it into the default installation directory. Any change to the system configuration done before the upgrade could be overwritten, so care has to be taken for the previous configuration to be backed up and reinstated. To prevent inadvertent overwrite of an existing configuration file, if a file `plantconf.cfg` is found in the installation directory, a popup dialog appears allowing to choose the course of action.



Warning: existing configuration!

A file plantconf.xml is already present in C:\Program Files\GrowthScheduler
What do you want to do?

- Keep the existing file
- Load a saved file from another path
- Replace the file with the default file provided by the installer

The current installer also writes a file `GrowthPaths.ini` in the application directory (see 3.7; this is also changed back and forth between versions, preferring alternatively the User Application Data directory). This file is overwritten at every installation.

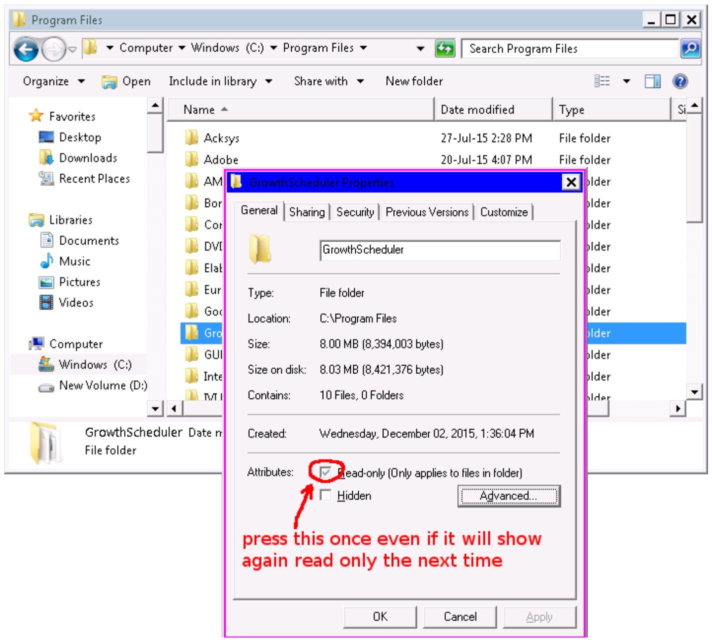
## 2.3 Note about Windows file permissions

Depending on the machine and on the user account performing the installation, the files in the Application Directory may result as write protected or not for the ordinary user. This is a hindrance, because `plantconf.cfg` and `GrowthPaths.ini`

[1]Occasionally, due to some automatic Labview builder numbering of versions, and when I have used different development machines to build different installers, the operating system may misjudge the order of versions. In such case too, uninstallation first is needed.

need to be rewritten by the program, the former from time to time, and the second at the end of every session. In case the program is found to be not able to rewrite them, these are a couple of points to check:

- the application directory (or perhaps just the relevant files) should be writable



| | | |
|---|---|---|
| Acksys | 27-Jul-15 2:28 PM | File folder |
| Adobe | 20-Jul-15 4:07 PM | File folder |

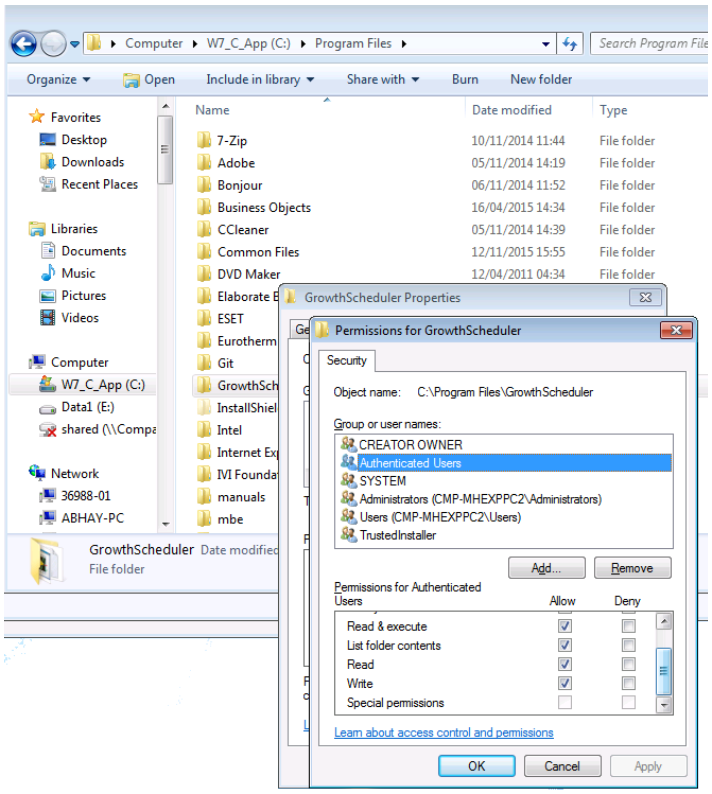
GrowthScheduler

Type: File folder
Location: C:\Program Files
Size: 8.00 MB (8,394,003 bytes)
Size on disk: 8.03 MB (8,421,376 bytes)
Contains: 10 Files, 0 Folders
Created: Wednesday, December 02, 2015, 1:36:04 PM
Attributes: [✓ Read-only (Only applies to files in folder)] Hidden

press this once even if it will show again read only the next time

GrowthScheduler File folder

- There should be writing privileges for a group “Authenticated Users”:



| Name | Date modified | Type |
|---|---|---|
| 7-Zip | 10/11/2014 11:44 | File folder |
| Adobe | 05/11/2014 14:19 | File folder |
| Bonjour | 06/11/2014 11:52 | File folder |
| Business Objects | 16/04/2015 14:34 | File folder |
| CCleaner | 05/11/2014 14:39 | File folder |
| Common Files | 12/11/2015 15:55 | File folder |
| DVD Maker | 12/04/2011 04:34 | File folder |

Object name: C:\Program Files\GrowthScheduler

Group or user names:
CREATOR OWNER
Authenticated Users
SYSTEM
Administrators (CMP-MHEXPPC2\Administrators)
Users (CMP-MHEXPPC2\Users)
TrustedInstaller

| Permissions for Authenticated Users | Allow | Deny |
|---|---|---|
| Read & execute | ✓ | |
| List folder contents | ✓ | |
| Read | ✓ | |
| Write | ✓ | |
| Special permissions | | |

If a rule for this group is not listed, it has to be created.

## 2.4 Multiple installations

It could be convenient to have more than a single copy of the program installed on a computer. This has to be done manually, duplicating the Application Folder, and thus is not tracked by the OS.

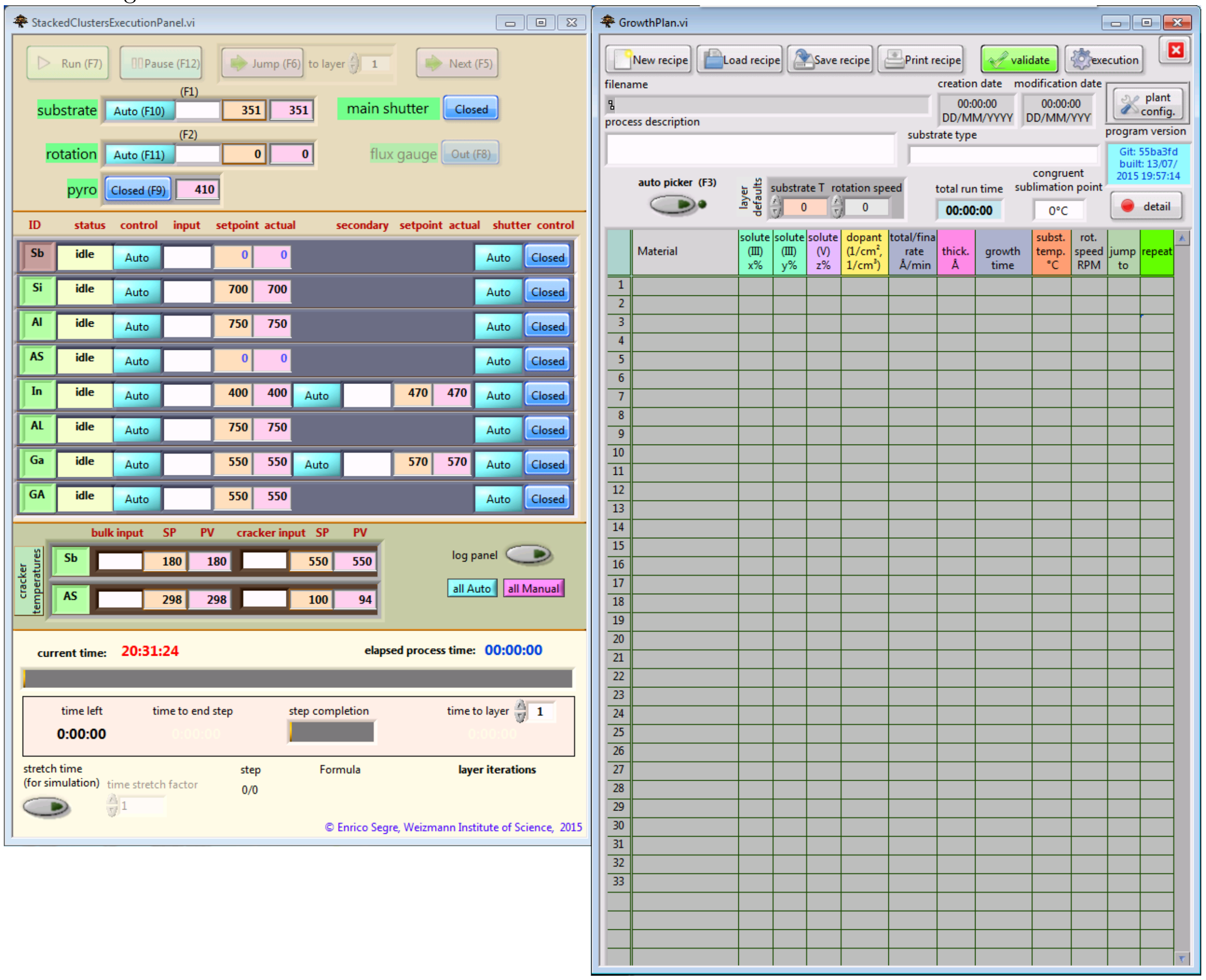
One case use for multiple installations is convenience of reviewing independently (offline) recipes and simulate the execution of different plans in different copies of the program. There could be other strategies for that and we're not yet fixed on a definite scheme, see discussion in 7.

# Chapter 3

# Operation

## 3.1 Program overview

At startup, two main windows open: the **Growth Recipe** and the **Plant control** panels. Both are described in detail in the following.



substrate (F1) Auto (F10) 351 351
rotation (F2) Auto (F11) 0 0
pyro Closed (F9) 410
main shutter Closed
flux gauge Out (F8)

| ID | status | control | input | setpoint | actual | secondary | input | setpoint | actual | shutter | control |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sb | idle | Auto | | 0 | 0 | | | | | Auto | Closed |
| Si | idle | Auto | | 700 | 700 | | | | | Auto | Closed |
| Al | idle | Auto | | 750 | 750 | | | | | Auto | Closed |
| AS | idle | Auto | | 0 | 0 | | | | | Auto | Closed |
| In | idle | Auto | | 400 | 400 | Auto | | 470 | 470 | Auto | Closed |
| AL | idle | Auto | | 750 | 750 | | | | | Auto | Closed |
| Ga | idle | Auto | | 550 | 550 | Auto | | 570 | 570 | Auto | Closed |
| GA | idle | Auto | | 550 | 550 | | | | | Auto | Closed |

| | bulk input | SP | PV | cracker input | SP | PV |
|---|---|---|---|---|---|---|
| Sb | | 180 | 180 | | 550 | 550 |
| AS | | 298 | 298 | | 100 | 94 |

current time: 20:31:24
elapsed process time: 00:00:00
time left 0:00:00
time to end step
step completion
time to layer 1
stretch time (for simulation)
time stretch factor 1
step 0/0
Formula
layer iterations

filename
creation date 00:00:00 DD/MM/YYYY
modification date 00:00:00 DD/MM/YYYY
process description
substrate type
program version Git: 55ba3fd built: 13/07/2015 19:57:14
auto picker (F3)
substrate T 0
rotation speed 0
total run time 00:00:00
congruent sublimation point 0°C

The **Growth recipe** panel opens with empty entries. The **Plant control** panel is shaped according to the previously saved hardware configuration. That is, it may have a different appearance than in figure the first time, and adapt once the system is configured as described in the following (3.2). The **Plant control** panel monitors, live from the moment it opens, readouts from the configured hardware, if present, or fake data, for simulation purposes. It can even be closed, and retrieved again with the button “Execution” in the **Growth recipe** panel, without interruption of its full running state. Closing the **Growth recipe** panel instead quits the program.

The casual user of an already configured plant, not interested in the details of the configuration process, would probably skip from this point to 3.3.

## 3.2 Hardware configuration

In the intended workflow, the hardware configuration panel is seldom accessed, as it needs to be set up only once for each plant, and changed only when cells are exchanged or recommissioned; periodically the effusion performance may be measured and the rate configuration may need to be corrected, but, all together, not as part of the everyday procedure.

The configuration panel is modal, to prevent any other program action while the configuration may be changed. It is opened with the dedicated button on the **Growth recipe** panel.

The configuration panel has two tabs, one for defining everything related to the effusion cells (3.2.1), and the other for configuring about everything else, that is properties of hardware in the main growth chamber, and general readout intervals and timings (3.2.2).

### 3.2.1 Cell Configuration

The left stack (refer to the figures shown in the following paragraphs) defines general properties of each effusion cell. New entries are created by typing into the fields of the next-to last element of the array, or using the right-click contextual menu “Insert element” at any point of it.[1]

Each cell has the following global properties:

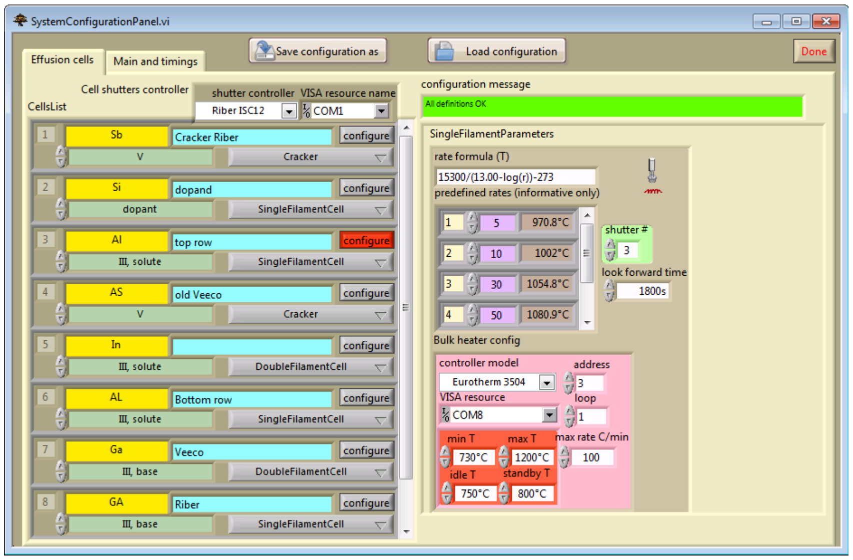
- a short identifier (yellow field), which is used throughout the Material formulas (3.3.1.1); commonly, chemical element symbols are used;
- a long, free text description (light blue)
- an elemental type property, which can be III base, III solute, (IV) dopant, V (the roman number being the chemical valence group in the periodic table). The correct type rules how the element can be used in growth formulas.
- a cell kind, currently Single Filament, Double Filament or Cracker. Different cell kinds have different hardware components and need a specific different configuration subpanel.[2]

Pressing the configure button, the specific configuration panel opens on the right. The button of the cell currently configured is highlighted in red.

In all systems we thought of, all cell shutters are driven by a single multishutter controller, which may or may not be responsible for further controls like main cell shutters and actuators. The controls for setting the communication resource and the device type of the shutter controller are placed above the cell list, the individual channel for each cell is set in its particular configuration subpanel.

[1]Currently, Insert element resets all the detailed configuration of the following cells, see related ticket
[2]Currently, changing the cell kind resets all the detailed configuration of the cell, see related ticket

#### 3.2.1.1 Single Filament effusion cells



Cell shutters controller — shutter controller: Riber ISC12; VISA resource name: COM1

CellsList

| | | | | |
|---|---|---|---|---|
| 1 | Sb | Cracker Riber | V | Cracker |
| 2 | Si | dopand | dopant | SingleFilamentCell |
| 3 | Al | top row | III, solute | SingleFilamentCell |
| 4 | AS | old Veeco | V | Cracker |
| 5 | In | | III, solute | DoubleFilamentCell |
| 6 | AL | Bottom row | III, solute | SingleFilamentCell |
| 7 | Ga | Veeco | III, base | DoubleFilamentCell |
| 8 | GA | Riber | III, base | SingleFilamentCell |

configuration message: All definitions OK

SingleFilamentParameters

rate formula (T): 15300/(13.00-log(r))-273

predefined rates (informative only)

| | | |
|---|---|---|
| 1 | 5 | 970.8°C |
| 2 | 10 | 1002°C |
| 3 | 30 | 1054.8°C |
| 4 | 50 | 1080.9°C |

shutter #: 3

look forward time: 1800s

Bulk heater config

controller model: Eurotherm 3504; address: 3

VISA resource: COM8; loop: 1

min T: 730°C; max T: 1200°C; max rate C/min: 100

idle T: 750°C; standby T: 800°C

A single filament cell has a bulk heater, whose temperature is controlled according to the effusion rate (and hence the epitaxial growth rate) desired. The cells are periodically calibrated to assess the $R(T)$ dependence, which is usually assumed to be in the form $R = a\,e^{-b/T}$. The inverse formula $T(R)$ is input. For generality, it is not limited to the form $T = A/(B - \log R)$; other expressions can be input. The syntax is that of the LabVIEW formula parser.

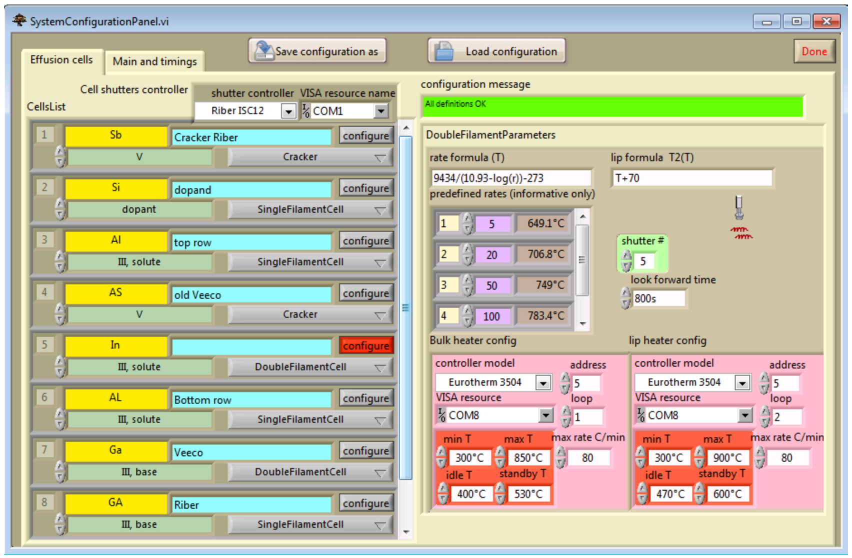
It is customary to write growth formulas as described in 3.3.1.1, where the element name is postpended with a suffix indicating a predefined rate. This suffix is free text, usually one figure numbers are used. A table of predefined labels and rates is filled, where the resulting bulk temperature is automatically computed.

The predefined rates are binding if the cell is used for a base III element or for a dopant, but are only informative for a solute III, because in practice the rate of a solute III is computed and not a primary input.

The *Look Forward Time* is used during process scheduling to allow preheating and stabilization of cells before their use in the process. The logic is that a cell setpoint must be commanded at least $LFT$ seconds before the cell is first used; and if the growth process is programmed with intermittent uses of a cell, the cell is brought to a *standby* state if the pause is longer than $2\,LFT$. (If a process is programmed with changes of the setpoint within a time shorter that $LFT$ between intermittent uses, the setpoint is changed as soon as possible, and it is the responsibility of the writer of growth recipes to run judiciously into such case).

For the heater itself, the communication address has to be defined; moreover, the range temperatures and the temperature at which the cell is to be held at idle time (*idle T*) and while standing between discontinuous uses in the course of a process (*standby T*). At the moment the choice of two different Eurotherm controllers for the heaters is offered; however, the communication to them is identical and coded as a single case. The controller list might be expanded in future if the need arises.

#### 3.2.1.2 Double Filament effusion cells



SystemConfigurationPanel.vi

Effusion cells | Main and timings | Save configuration as | Load configuration | Done

Cell shutters controller — shutter controller: Riber ISC12; VISA resource name: COM1

CellsList

| | | | |
|---|---|---|---|
| 1 | Sb | Cracker Riber | configure |
| | V | Cracker | |
| 2 | Si | dopand | configure |
| | dopant | SingleFilamentCell | |
| 3 | Al | top row | configure |
| | III, solute | SingleFilamentCell | |
| 4 | AS | old Veeco | configure |
| | V | Cracker | |
| 5 | In | | configure |
| | III, solute | DoubleFilamentCell | |
| 6 | AL | Bottom row | configure |
| | III, solute | SingleFilamentCell | |
| 7 | Ga | Veeco | configure |
| | III, base | DoubleFilamentCell | |
| 8 | GA | Riber | configure |
| | III, base | SingleFilamentCell | |

configuration message: All definitions OK

DoubleFilamentParameters

rate formula (T): 9434/(10.93-log(r))-273

lip formula T2(T): T+70

predefined rates (informative only)

| | | |
|---|---|---|
| 1 | 5 | 649.1°C |
| 2 | 20 | 706.8°C |
| 3 | 50 | 749°C |
| 4 | 100 | 783.4°C |

shutter #: 5

look forward time: 800s

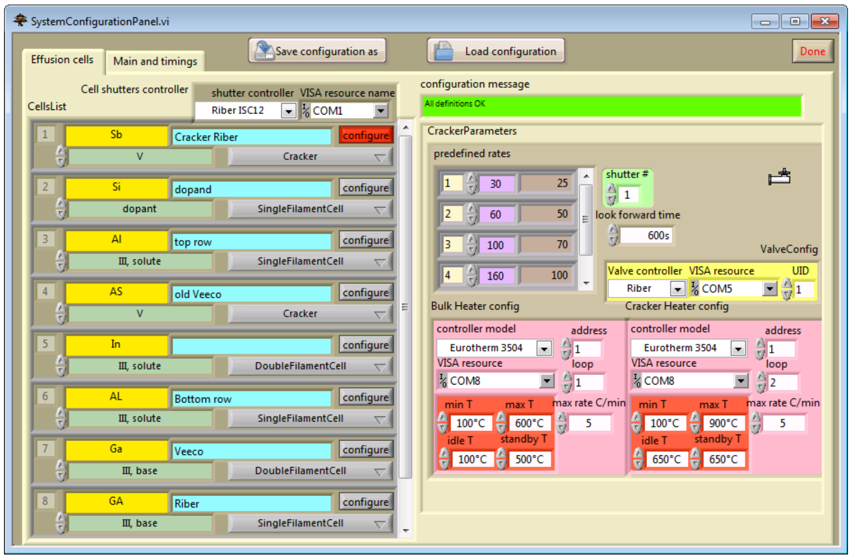
| | Bulk heater config | lip heater config |
|---|---|---|
| controller model | Eurotherm 3504 | Eurotherm 3504 |
| address | 5 | 5 |
| VISA resource | COM8 | COM8 |
| loop | 1 | 2 |
| min T | 300°C | 300°C |
| max T | 850°C | 900°C |
| max rate C/min | 80 | 80 |
| idle T | 400°C | 470°C |
| standby T | 530°C | 600°C |

Double filament cells are analogous to single filament cells, with the presence of an additional heater for the lip, having the same configuration parameters of the bulk heater. During growths the lip temperature is set as a function of the bulk temperature, as expressed by the *lip formula*.

#### 3.2.1.3 Cracker effusion cells

The effusion rate of a cracker is determined by a valve position, and not by the temperatures of the heaters. The predefined rates are the breakpoints for linear interpolation, when an arbitrary rate is used.

The valve driver and communication address is chosen in the apposite subpanel. Implemented drivers at the present time are for the older Veeco SMC (bisync serial), for the newer Veeco SMC-III Flexcomm (modbus serial, as we haven't been able to use bisync serial, theoretically identical to the former), and Riber AVP (modbus serial). A further option "manual" is provided out of no other choice. When that is chosen, the communication resource chosen is irrelevant.



SystemConfigurationPanel.vi

Effusion cells | Main and timings | Save configuration as | Load configuration | Done

Cell shutters controller — shutter controller: Riber ISC12; VISA resource name: COM1

CellsList

| | | | |
|---|---|---|---|
| 1 | Sb | Cracker Riber | configure |
| | V | Cracker | |
| 2 | Si | dopand | configure |
| | dopant | SingleFilamentCell | |
| 3 | Al | top row | configure |
| | III, solute | SingleFilamentCell | |
| 4 | AS | old Veeco | configure |
| | V | Cracker | |
| 5 | In | | configure |
| | III, solute | DoubleFilamentCell | |
| 6 | AL | Bottom row | configure |
| | III, solute | SingleFilamentCell | |
| 7 | Ga | Veeco | configure |
| | III, base | DoubleFilamentCell | |
| 8 | GA | Riber | configure |
| | III, base | SingleFilamentCell | |

configuration message: All definitions OK

CrackerParameters

predefined rates

| | | |
|---|---|---|
| 1 | 30 | 25 |
| 2 | 60 | 50 |
| 3 | 100 | 70 |
| 4 | 160 | 100 |

shutter #: 1

look forward time: 600s

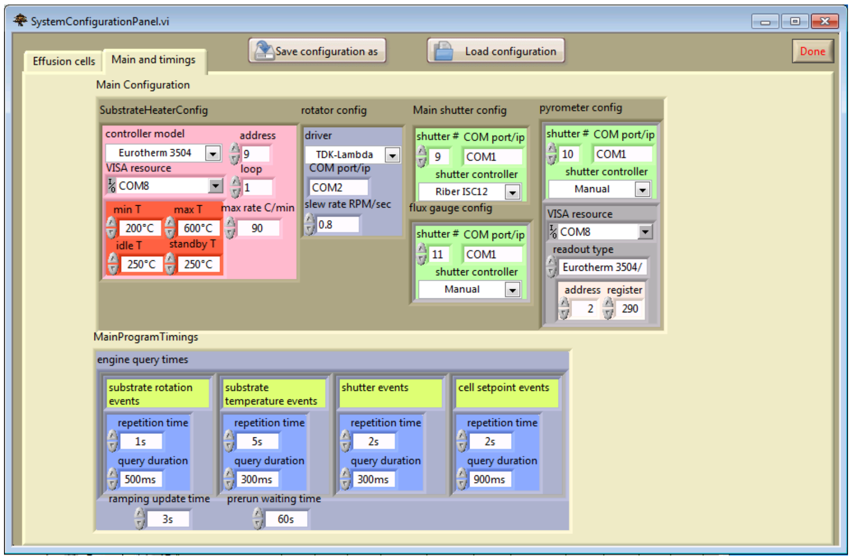
ValveConfig — Valve controller: Riber; VISA resource: COM5; UID: 1

| | Bulk Heater config | Cracker Heater config |
|---|---|---|
| controller model | Eurotherm 3504 | Eurotherm 3504 |
| address | 1 | 1 |
| VISA resource | COM8 | COM8 |
| loop | 1 | 2 |
| min T | 100°C | 100°C |
| max T | 600°C | 900°C |
| max rate C/min | 5 | 5 |
| idle T | 100°C | 650°C |
| standby T | 500°C | 650°C |

The look forward time is taken into account to change anticipatedly the valve setpoint, but enters otherwise there is no setting of valves in *standby* or *idle* positions.

Crackers have also two heaters, whose addresses and temperature ranges have to be set. Differently than other cells, however, these temperatures can only be controlled manually in the execution panel and are not driven by the growth process. Therefore, while the min and max temperatures have a safety meaning, the idle and standby temperatures are irrelevant. For programming economy, the same heater configuration template has been used as for the other cells.

### 3.2.2 Main and timing configuration



Main Configuration

SubstrateHeaterConfig
controller model: Eurotherm 3504
address: 9
VISA resource: COM8
loop: 1
min T: 200°C
max T: 600°C
max rate C/min: 90
idle T: 250°C
standby T: 250°C

rotator config
driver: TDK-Lambda
COM port/ip: COM2
slew rate RPM/sec: 0.8

Main shutter config
shutter #: 9
COM port/ip: COM1
shutter controller: Riber ISC12

flux gauge config
shutter #: 11
COM port/ip: COM1
shutter controller: Manual

pyrometer config
shutter #: 10
COM port/ip: COM1
shutter controller: Manual
VISA resource: COM8
readout type: Eurotherm 3504/
address: 2
register: 290

MainProgramTimings

engine query times

| | substrate rotation events | substrate temperature events | shutter events | cell setpoint events |
|---|---|---|---|---|
| repetition time | 1s | 5s | 2s | 2s |
| query duration | 500ms | 300ms | 300ms | 900ms |

ramping update time: 3s
prerun waiting time: 60s

The subframes of this tab have the following function:

**Substrate Heater** has the same configuration mask as the cell heaters, for ~~lazy programming~~ uniformity. The idle temperature is that at which the substrate is sent at the end of a growth process; the standby temperatures is irrelevant.

**Substrate rotator** besides specifying the communication port and the type of the controller, the slew rate specifies the maximal angular acceleration/deceleration, which the user may want to keep low in order to prevent unnecessary impulsive stresses and friction.

**Pyrometer** as one of the possible ways of reading it out has hacked to be using one Eurotherm A/D channel, the address of that can be specified.

If the *controller* or *driver* field of one of these devices is set to Manual (for shutters) or Unknown (pyrometer readout) or custom (rotator), the communication parameters are ignored, and there is no risk of conflict with other devices.

**Engine query times** state variables are periodically read, unless a write to the same controller is imminent. These control allow to specify the relevant timings: with which frequency to read (the more, the more immediate indication in the Plant Control Panel 3.5), and how much time each read is suppose to take at most (to cancel a read if a scheduled write is supposed to be sent before that time). There are presently four of such timing controls, as the choice has been to use four engines (5).

**Ramping update time** How often the temperature of a ramping cell is recalculated. As the heaters are slow responding objects, it doesn't make exceeding sense to send temperature change commands more than once in every few seconds.

**Prerun waiting time** The guard time before starting a process, as described in 3.5.2.1.

### 3.2.3 Configuration files

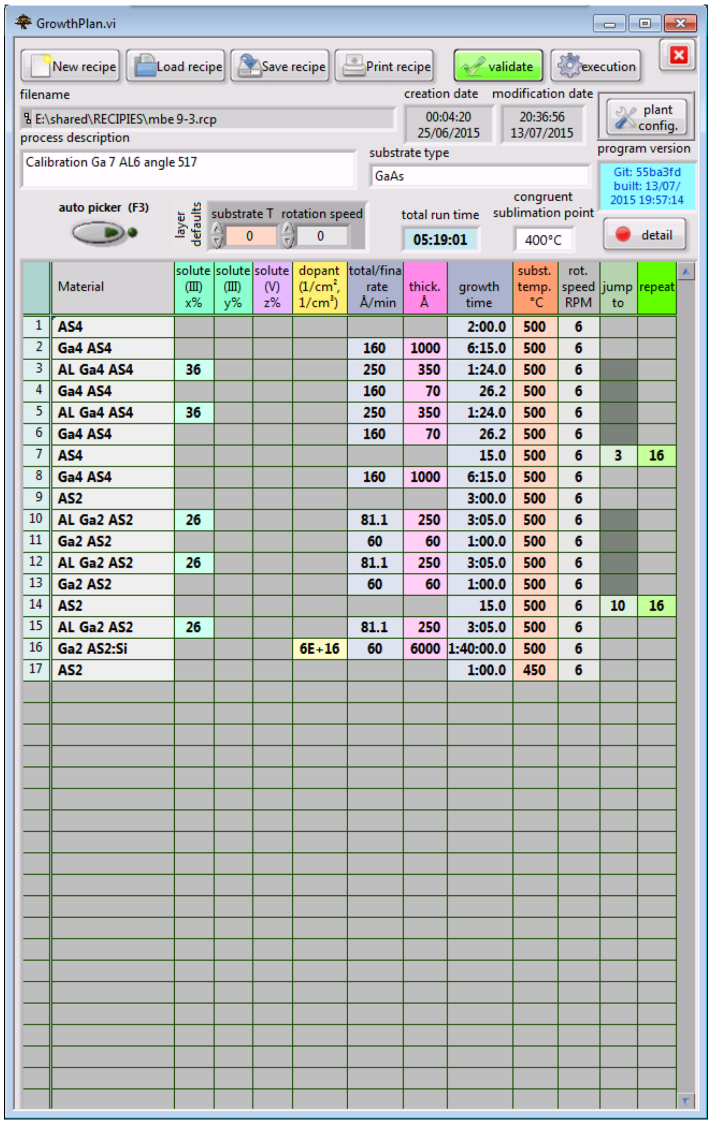
On closing the panel, all configuration changes are effected in the various parts of the program (fact which may invalidate the current recipe or settings), and the new configuration is written in to the default file `plantconf.cfg`. The current logic is the following. It can perhaps be changed in future, it has some drawbacks but it also its rationale and advantage.

- At program start, the default configuration is read from the file `plantconf.cfg`, which resides in the same directory the same directory of **GrowthScheduler.exe** (or in that of **GrowthPlan.vi**, if the sources are tested in the Labview IDE).

- Every time the configuration panel is closed, the file `plantconf.cfg` is overwritten, either that the configuration really changed or not.
- The configuration panel has in any event buttons for a) saving the current configuration under any other name in any directory; b) load a configuration file with arbitrary pathname.
- It is useful and recommendable to keep safety copies of configurations at a given instant in time. To that extent the principal file `plantconf.cfg` can be copied around wit OS action in a safe place, with the same or with a different name; or the configuration panel can be used to save it under another name.
- When a configuration file is loaded with the configuration panel, the new configuration is saved in `plantconf.cfg`.

As a matter of fact, the file `plantconf.cfg` is not meant to be easily edited as text file, externally by the user. Some reasoning is in 4.1.1.

## 3.3 Growth recipe panel



filename: E:\shared\RECIPIES\mbe 9-3.rcp
creation date: 00:04:20 25/06/2015
modification date: 20:36:56 13/07/2015
process description: Calibration Ga 7 AL6 angle 517
substrate type: GaAs
program version: Git: 55ba3fd built: 13/07/2015 19:57:14
layer defaults: substrate T 0, rotation speed 0
total run time: 05:19:01
congruent sublimation point: 400°C

| | Material | solute (III) x% | solute (III) y% | solute (V) z% | dopant ($1/cm^2$, $1/cm^3$) | total/fina rate Å/min | thick. Å | growth time | subst. temp. °C | rot. speed RPM | jump to | repeat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | AS4 | | | | | | | 2:00.0 | 500 | 6 | | |
| 2 | Ga4 AS4 | | | | | 160 | 1000 | 6:15.0 | 500 | 6 | | |
| 3 | AL Ga4 AS4 | 36 | | | | 250 | 350 | 1:24.0 | 500 | 6 | | |
| 4 | Ga4 AS4 | | | | | 160 | 70 | 26.2 | 500 | 6 | | |
| 5 | AL Ga4 AS4 | 36 | | | | 250 | 350 | 1:24.0 | 500 | 6 | | |
| 6 | Ga4 AS4 | | | | | 160 | 70 | 26.2 | 500 | 6 | | |
| 7 | AS4 | | | | | | | 15.0 | 500 | 6 | 3 | 16 |
| 8 | Ga4 AS4 | | | | | 160 | 1000 | 6:15.0 | 500 | 6 | | |
| 9 | AS2 | | | | | | | 3:00.0 | 500 | 6 | | |
| 10 | AL Ga2 AS2 | 26 | | | | 81.1 | 250 | 3:05.0 | 500 | 6 | | |
| 11 | Ga2 AS2 | | | | | 60 | 60 | 1:00.0 | 500 | 6 | | |
| 12 | AL Ga2 AS2 | 26 | | | | 81.1 | 250 | 3:05.0 | 500 | 6 | | |
| 13 | Ga2 AS2 | | | | | 60 | 60 | 1:00.0 | 500 | 6 | | |
| 14 | AS2 | | | | | | | 15.0 | 500 | 6 | 10 | 16 |
| 15 | AL Ga2 AS2 | 26 | | | | 81.1 | 250 | 3:05.0 | 500 | 6 | | |
| 16 | Ga2 AS2:Si | | | | 6E+16 | 60 | 6000 | 1:40:00.0 | 500 | 6 | | |
| 17 | AS2 | | | | | | | 1:00.0 | 450 | 6 | | |

In this panel, growth process recipes can be edited, loaded, saved. The main control of this panel is the table containing the growth recipe itself. In addition, the buttons for (re)opening the **Configuration** panel (described in 3.2.1), the **Graphs and Details** panel (3.4) and the **Plant control** panel (3.5) are located here.

Entries of the table input by the user are displayed in black font, derived values computed according to the rules are printed in violet. The content of a layer row is validated as it is typed in, marking with a red background missing or invalid entries, and computing derived values as data is available. Input focus is automated so that as soon as one value is typed and Enter/Return are pressed, focus is moved to the next relevant input cell. The resulting total growth time is also updated as it goes.

### 3.3.1 The Recipe: Layer notation

The program is devised for the growth of epitaxial layers of varying composition and thickness, substantially based on heterogeneous III-V compounds, with dopants. Each row of the growth table reflects a stage of this process, either a growth one, during which effusion cells are brought to prescribed temperatures and respective shutters are open, or growth pauses, during which only element V cells are operated, keeping the effusion valves open, in order to maintain a supporting atmosphere.

#### 3.3.1.1 Layer materials and associated input

The following different compositions are admitted. The one or the other determines which other entries in the layer row are meaningful, and whether they are automatically computed or need to be input. In the following, **A** stands for a solute group III component, **B** for a base element of group III, **C** for a group V, and **D** for a dopant. Valid possibilities are:

1. **B C** binary layer with base III.
   Only the layer thickness is input. The growth rate of the layer is determined by the predefined rate chosen for **B**.

2. **A B C** ternary solid solution.
   Inputs are the percentage $x\%$ of **A**, and the layer thickness. The component **B** is present at $(100 - x)\%$.

3. **A A B C** quaternary solid solution.
   Inputs are the percentages $x\%$ and $y\%$ of the two solutes **A**, and the layer thickness. The component **B** is present at $(100 - x - y)\%$.

4. **A C** binary layer with 100% "solute" III.
   Inputs are the desired growth rate and the desired layer thickness.

5. **A A C** ternary solid solution without base III
   Inputs are the desired growth rate, the layer thickness and the percentage $x\%$. The fraction $y\% = 100 - x\%$ is filled in automatically.

6. **C, C C** growth interruption.
   The only input is the desired growth (pause) time. In the case **C C**, the column for the percentage $z\%$ can be filled so as to show an indicative value, but it doesn't affect any computation[3].

7. **rA B C**, **rA A C** ramped solid solutions (**rA rA C** and **rA rA B C** are also possible though I understand not used in practice)
   The concentrations of the ramped solutes start from the concentration of the corresponding component at the end of the previous layer, and vary linearly with height till a final value. The inputs are the same as for cases 2. and 5., but solution percentages input are intended as those to be reached at the end of the layer.

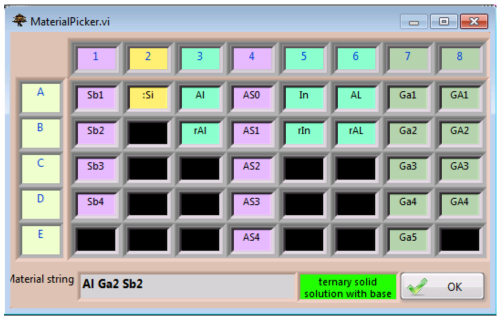
8. **B C:D**, **A B C:D**, **A A B C:D**, **A C:D**, **A A C:D**, 3-d doped layers.
   As the corresponding preceding cases, with the addition of the dopant. The additional input is the desired bulk dopant concentration, which is assumed not to affect the growth rate.

9. **C:D** 2-d doping ($\Delta$-doping).
   Inputs are the surface dopant concentration and growth time.

The convention is to display the formula with its constituents in that order, i.e. **Al Ga3 As3** for case 2, **rAl Ga4 As4** for case 7, etc.

#### 3.3.1.2 Material Picker

To avoid mistakes in inserting formulas, a modal popup picker is provided. The picker opens automatically whenever focus is in a Material cell in the Growth table, if Auto Picker is on; otherwise it can be invoked with the binding F3.



|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | Sb1 | :Si | Al | AS0 | In | AL | Ga1 | GA1 |
| B | Sb2 | ■ | rAl | AS1 | rIn | rAL | Ga2 | GA2 |
| C | Sb3 | ■ | ■ | AS2 | ■ | ■ | Ga3 | GA3 |
| D | Sb4 | ■ | ■ | AS3 | ■ | ■ | Ga4 | GA4 |
| E | ■ | ■ | ■ | AS4 | ■ | ■ | Ga5 | ■ |

Material string: Al Ga2 Sb2

ternary solid solution with base

The buttons of the picker are generated from what defined in the cell configuration. The color of the buttons reflects the role of the component: olive green for base group III components, light green for solute III, yellow for dopants, fuchsia for group V. Components are chosen either pressing on the buttons, or typing the letter and the figure corresponding to the desired cell and rate, e.g. typing **C7** or **7C** in the example shown in figure to obtain **Ga3** in the formula. Components can be entered in any order, the picker validates and normalizes the formula as it goes.

To replace a component previously entered with a given rate, with a different rate (e.g. **Ga1** with **Ga2**), it is sufficient to press the button for the new rate, or to type the new cross-code.

To remove a component from a formula, it is possible to press on a black empty cell in the relative column.

[3] I wonder if $z\%$ shouldn't result instead from the rates chosen, which are in flux units anyway.

**Picker keyboard shortcuts**

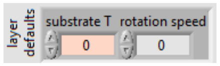
**Return** exits the picker, sending the constructed formula to the current cell in the growth table

**Escape** exits the picker, without passing the modified material

**Backspace** removes the last component entered

**Delete** clears the current formula

#### 3.3.1.3 Substrate rotation and temperature



They have to be filled in for every layer. However, the default values written in advance into “layer defaults” are proposed, as a convenience.

#### 3.3.1.4 Looping (periodic structures, superlattices)

If a given set of consecutive layers is to be repeated a number of times, the cell “jump to” of the last layer of the set is to be filled with the number of the first layer of the set, and the “repeat” cell with the number of repetitions. All the “jump to” cells of the member layers are highlighted in dark grey to mark the stretch. In MBE language, such repetition is said to form a superlattice. A recipe can contain more than one non-overlapping repetition. Nested loops are allowed, even if not used in practice.

#### 3.3.1.5 Keyboard shortcuts in the Growth Table

**F3** on a Material cell, pop up the Material picker (unless the switch “auto picker” is toggled)

**Enter, Shift-Right** move to the next editable cell on row, or to the beginning of the next row

**Shift-Enter, Shift-Left** move to the previous editable cell

**Shift-Up** move to the editable cell directly above, or to the first editable cell on the right in the row above the current, if the cell directly above is not editable

**Shift-Down** move to the editable cell directly below, or to the first editable cell on the right in the row below the current, if the cell directly below is not editable

**Shift-Insert, Ctrl-Insert** insert a layer low at the before the current position

**Shift-Delete, Ctrl-Delete** remove the current layer row

### 3.3.2 Auxiliary recipe information

These fields are saved along with the recipe:

**Process description** free text

**Substrate type** free text

**Congruent sublimation point** temperature in °C. The grown crystal has to be maintained under element V atmosphere as long as the substrate temperature is higher than this value, as it goes in 3.5.2.6.

### 3.3.3 Load, save, print recipe

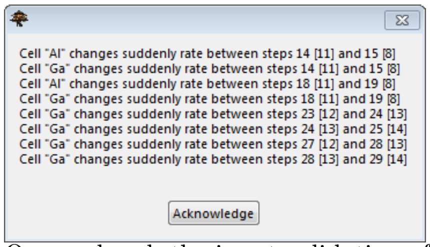
By self-explanatory buttonry.

Recipe files themselves are tab-separated text files, plainly reflecting the process information and Growth table content, as input. The default extension of recipe files is `.rcp`.

When an existing recipe is loaded, its modification date is set to the current time.

For flexibility and device independence, Print generates a temporary html file with a colorized rendering of the recipe, and opens it in the default browser. From there, it can be printed or saved using the browser’s own commands.

### 3.3.4 Process compilation and verification

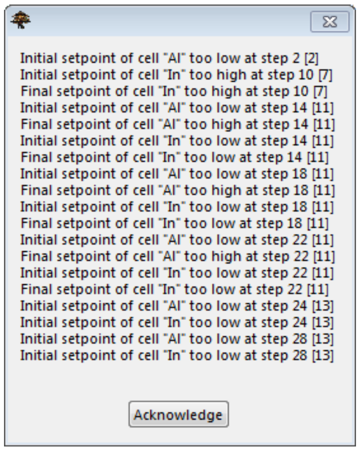


```
Cell "Al" changes suddenly rate between steps 14 [11] and 15 [8]
Cell "Ga" changes suddenly rate between steps 14 [11] and 15 [8]
Cell "Al" changes suddenly rate between steps 18 [11] and 19 [8]
Cell "Ga" changes suddenly rate between steps 18 [11] and 19 [8]
Cell "Ga" changes suddenly rate between steps 23 [12] and 24 [13]
Cell "Ga" changes suddenly rate between steps 24 [13] and 25 [14]
Cell "Ga" changes suddenly rate between steps 27 [12] and 28 [13]
Cell "Ga" changes suddenly rate between steps 28 [13] and 29 [14]
```

On one hand, the input validation of the table rows doesn't guarantee that the complete growth process is feasible, because of constraints on the allowed temperatures of the cells and because sudden rate changes on heated effusion cells cannot take place. On the other, the table content is not yet in the form of a process schedule.

Compilation and process validation happen when the “validate” button is pressed, or when focus is brought to the **Plant control panel** (3.5) or to the **Graph and process details panel** (3.4). In the current program implementation compilation is not immediate, and moreover it is supposed to bring up warning dialogs like those in figure, for unrealizable recipes. Because of this, compilation is not launched automatically for each change in the table content.

The “validate” button is grey if the recipe has been modified but not validated, yellow while it is validated, green if it is compiled and valid and red if it is compiled and invalid. Invalid recipes can be executed, but the results may be not those prescribed by the recipe table–cells may not reach in time the temperatures and rates prescribed, temperatures may be clipped to the ranges configured for the cells, layer thicknesses may be different than what asked, etc.



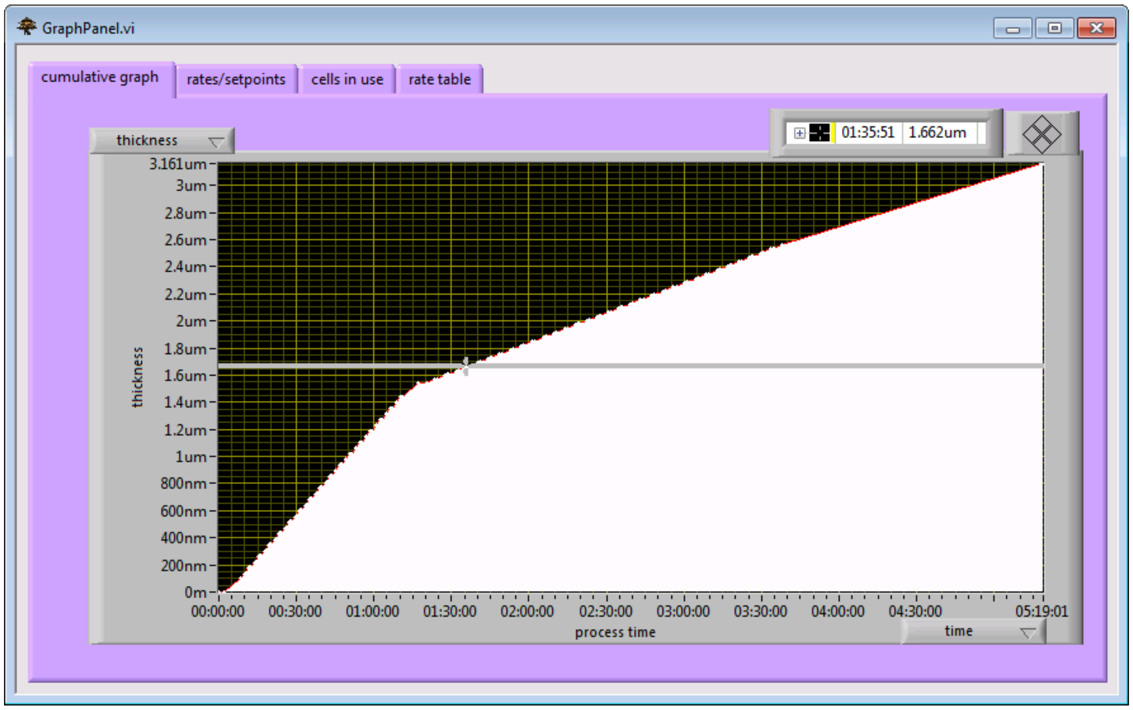
```
Initial setpoint of cell "Al" too low at step 2 [2]
Initial setpoint of cell "In" too high at step 10 [7]
Final setpoint of cell "In" too high at step 10 [7]
Initial setpoint of cell "Al" too low at step 14 [11]
Final setpoint of cell "Al" too high at step 14 [11]
Initial setpoint of cell "In" too low at step 14 [11]
Final setpoint of cell "In" too low at step 14 [11]
Initial setpoint of cell "Al" too low at step 18 [11]
Final setpoint of cell "Al" too high at step 18 [11]
Initial setpoint of cell "In" too low at step 18 [11]
Final setpoint of cell "In" too low at step 18 [11]
Initial setpoint of cell "Al" too low at step 22 [11]
Final setpoint of cell "Al" too high at step 22 [11]
Initial setpoint of cell "In" too low at step 22 [11]
Final setpoint of cell "In" too low at step 22 [11]
Initial setpoint of cell "Al" too low at step 24 [13]
Initial setpoint of cell "In" too low at step 24 [13]
Initial setpoint of cell "Al" too low at step 28 [13]
Initial setpoint of cell "In" too low at step 28 [13]
```

Layer repetitions (3.3.1.4) demand that the growth process is unrolled, to determine the correct antecedent and followers of a certain iteration of a given layer. This is reflected in reporting both the *step* number and the corresponding *[layer]* number, as displayed in the warning messages here and in the Rate table below (3.4.4).

## 3.4 Graphs and process details panel

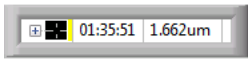
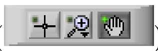
The button "Detail" on the **Growth recipe** panel opens a new window with four tabs. These tabs where initially conceived as a graphical verification aid during development, and haven't all been polished for clarity. All the information shown refers to the last recipe validated; however, opening or bringing the focus to this window triggers a validation of the recipe (if it was changed since), to actualize the content.

### 3.4.1 Cumulative thickness tab

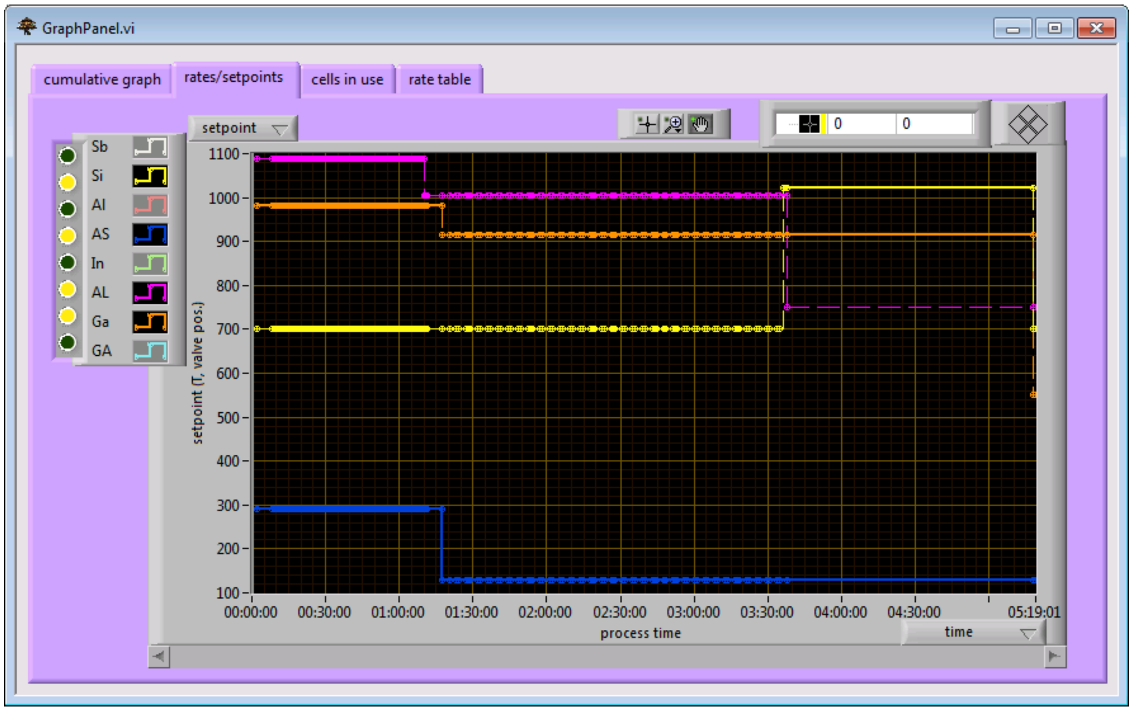


This would be a graph of the total growth thickness, or of the step timing as a function of either the growth time or the step number. It is not the most useful graph, nor it is accurate in the case of ramped layers, as it shows a stepwise linear approximation of the thickness rather than the real growth curve.

Some general interface features are common with the next two tabs:

- the data used as abscissa (e.g. step or growth time) and ordinata can be selected by pull-down menus by the axes;
- scale limits for abscissa and ordinata can be typed into, if the axes ane not set for Autoscale (which is changed by right click menu on them)
- A cursor can be dragged, to inspect the numerical values of the curve. This is a standard labview widget; the position of the cursor can also be set numerically by typing it in the cursor legend  01:35:51 1.662um (which needs to be opened, showing + on its left, clicking on that column)
- the cursor automatically follows the process, during execution;
- The cursor can be dragged only if the right graph tool is selected (). The middle tool allows zooming the graph by point and click, if Autoscale is unset.

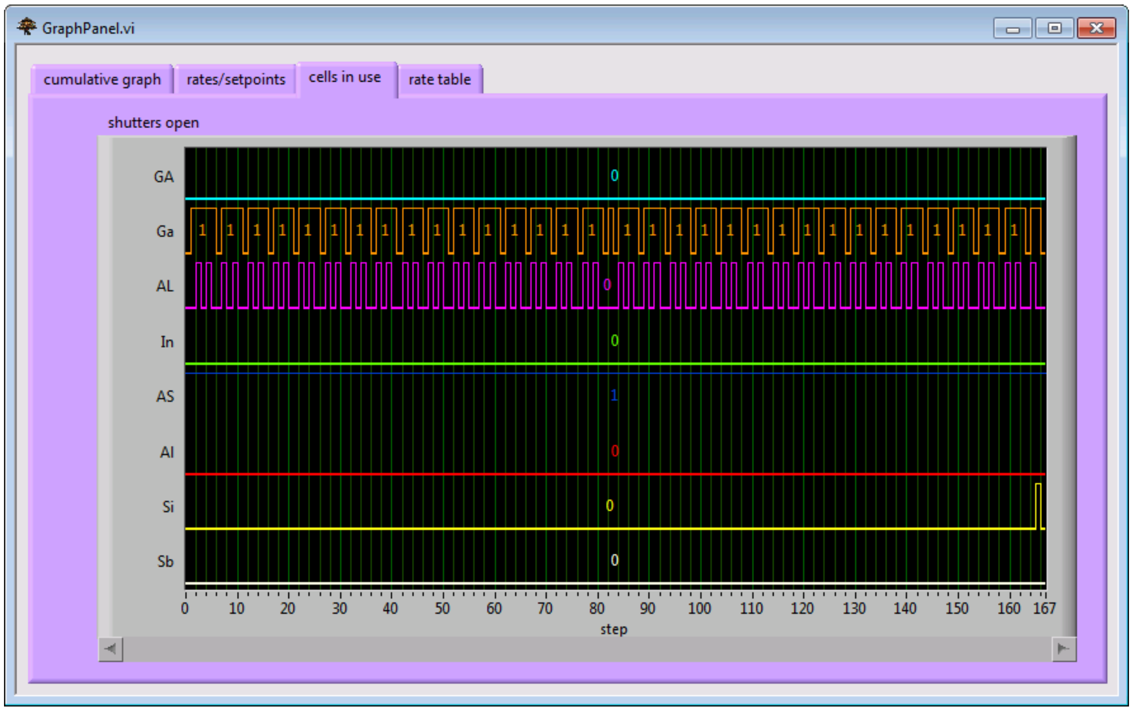
### 3.4.2 Setpoints and rates graph



This graph is a bit more useful. By default it shows only traces of the cells used by the last validated recipe; traces for any cell can be shown/hidden with the radio buttons on the left of the graph.

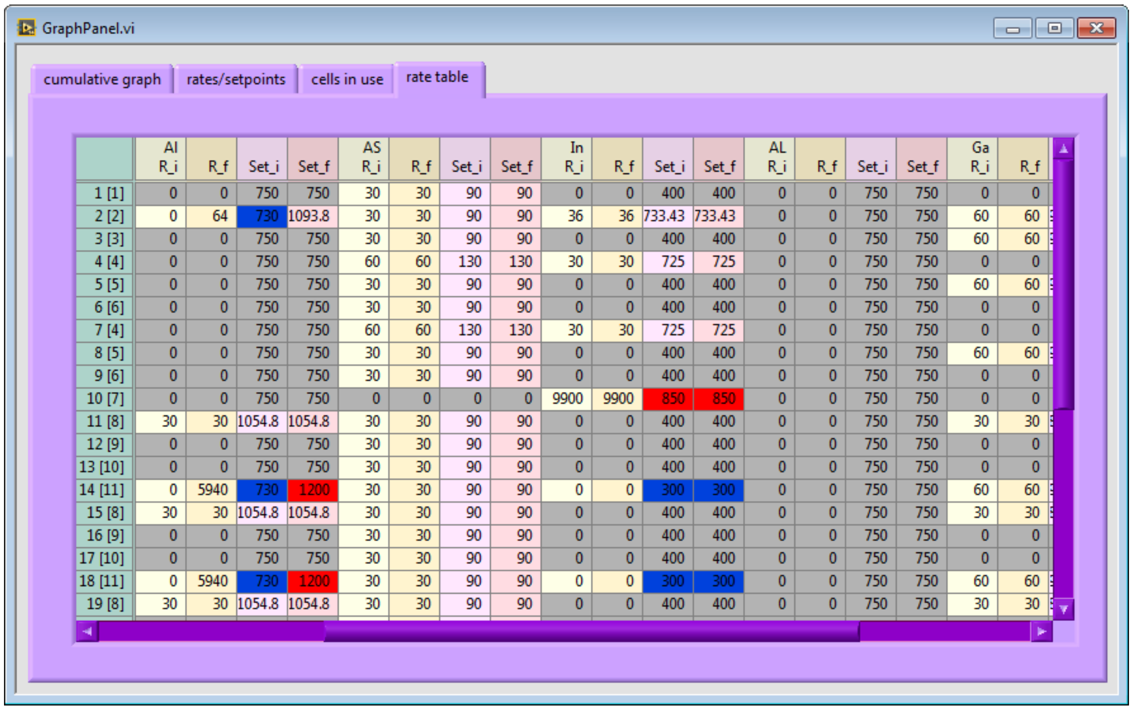
Continuous lines represent flux from cells momentarily in use, whereas dashed lines give the setpoint of equivalent rates at time the cell shutter is closed. Either the setpoint of the rate can change when a cell is not in use, if the schedule requires an anticipate preparation.

### 3.4.3 Cells use graph



This panel shows when cell shutters are closed/opened along the process as a digital graph, and is not particularly readable nor useful, admittedly.

### 3.4.4 Rate table

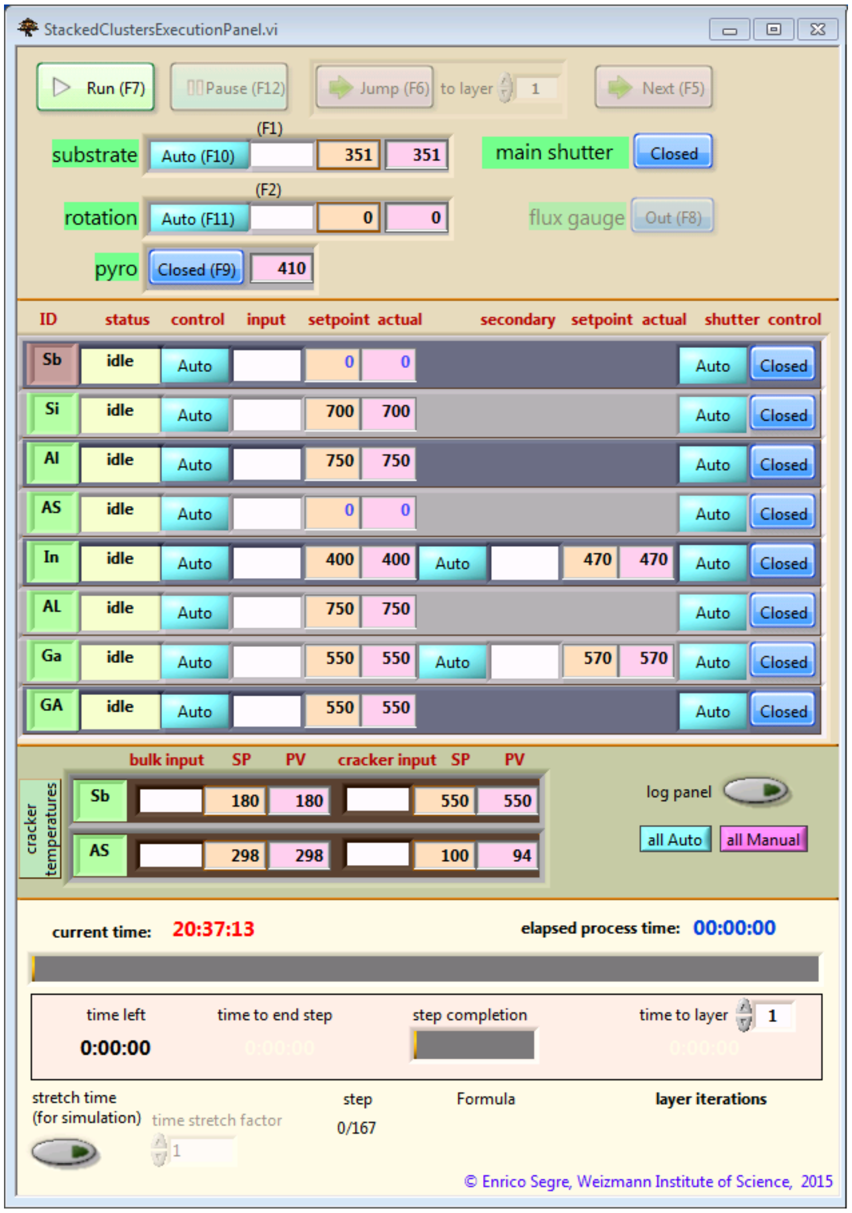


| | Al | | | | AS | | | | In | | | | AL | | | | Ga | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R_i | R_f | Set_i | Set_f | R_i | R_f | Set_i | Set_f | R_i | R_f | Set_i | Set_f | R_i | R_f | Set_i | Set_f | R_i | R_f |
| 1 [1] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 0 | 0 |
| 2 [2] | 0 | 64 | 730 | 1093.8 | 30 | 30 | 90 | 90 | 36 | 36 | 733.43 | 733.43 | 0 | 0 | 750 | 750 | 60 | 60 |
| 3 [3] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 60 | 60 |
| 4 [4] | 0 | 0 | 750 | 750 | 60 | 60 | 130 | 130 | 30 | 30 | 725 | 725 | 0 | 0 | 750 | 750 | 0 | 0 |
| 5 [5] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 60 | 60 |
| 6 [6] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 0 | 0 |
| 7 [4] | 0 | 0 | 750 | 750 | 60 | 60 | 130 | 130 | 30 | 30 | 725 | 725 | 0 | 0 | 750 | 750 | 0 | 0 |
| 8 [5] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 60 | 60 |
| 9 [6] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 0 | 0 |
| 10 [7] | 0 | 0 | 750 | 750 | 0 | 0 | 0 | 0 | 9900 | 9900 | 850 | 850 | 0 | 0 | 750 | 750 | 0 | 0 |
| 11 [8] | 30 | 30 | 1054.8 | 1054.8 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 30 | 30 |
| 12 [9] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 0 | 0 |
| 13 [10] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 0 | 0 |
| 14 [11] | 0 | 5940 | 730 | 1200 | 30 | 30 | 90 | 90 | 0 | 0 | 300 | 300 | 0 | 0 | 750 | 750 | 60 | 60 |
| 15 [8] | 30 | 30 | 1054.8 | 1054.8 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 30 | 30 |
| 16 [9] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 0 | 0 |
| 17 [10] | 0 | 0 | 750 | 750 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 0 | 0 |
| 18 [11] | 0 | 5940 | 730 | 1200 | 30 | 30 | 90 | 90 | 0 | 0 | 300 | 300 | 0 | 0 | 750 | 750 | 60 | 60 |
| 19 [8] | 30 | 30 | 1054.8 | 1054.8 | 30 | 30 | 90 | 90 | 0 | 0 | 400 | 400 | 0 | 0 | 750 | 750 | 30 | 30 |

This shows, step[layer] by step and cell by cell, the prescribed effusion rates $R_i$ at the beginning and $R_f$ at the end of the layer (the values may be different for ramped layers), as well as the cell setpoints $Set_i$ and $Set_f$, which can either be bulk temperatures for filament cells, or valve positions for crackers. The information displayed parallels the one graphically available in the Setpoints and rates graph (3.4.2), except for not showing the time, and the evolution in the course of a step.

Entries corresponding to cells unused in a given layer have a gray background; entries of cells used have cyclical pastel backgrounds, as a pure guide to the eye. Out of range temperatures are marked in blue if the request was too low and red if too high, and clipped to the acceptable range.

## 3.5 Plant control panel



substrate Auto (F10) (F1) 351 351 main shutter Closed

rotation Auto (F11) (F2) 0 0 flux gauge Out (F8)

pyro Closed (F9) 410

| ID | status | control | input | setpoint | actual | secondary | | setpoint | actual | shutter | control |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sb | idle | Auto | | 0 | 0 | | | | | Auto | Closed |
| Si | idle | Auto | | 700 | 700 | | | | | Auto | Closed |
| Al | idle | Auto | | 750 | 750 | | | | | Auto | Closed |
| AS | idle | Auto | | 0 | 0 | | | | | Auto | Closed |
| In | idle | Auto | | 400 | 400 | Auto | | 470 | 470 | Auto | Closed |
| AL | idle | Auto | | 750 | 750 | | | | | Auto | Closed |
| Ga | idle | Auto | | 550 | 550 | Auto | | 570 | 570 | Auto | Closed |
| GA | idle | Auto | | 550 | 550 | | | | | Auto | Closed |

cracker temperatures

| | bulk input | SP | PV | cracker input | SP | PV |
|---|---|---|---|---|---|---|
| Sb | | 180 | 180 | | 550 | 550 |
| AS | | 298 | 298 | | 100 | 94 |

current time: 20:37:13 elapsed process time: 00:00:00

time left 0:00:00 time to end step step completion time to layer 1

stretch time (for simulation) time stretch factor 1 step 0/167 Formula layer iterations

© Enrico Segre, Weizmann Institute of Science, 2015

Once a recipe is validated (3.3.4), the cells required are highlighted by a lighter gray background.

### 3.5.1 Manual control

#### 3.5.1.1 Keyboard shortcuts in the execution panel

In setpoint input fields (substrate rotaion, temperature, cell main and secondary setpoints, cracker temperatures), numbers can be typed and edited (e.g, the cursor is moved back and forth with Left/Right, text can be selected) before the input is set into effect.

**Enter, Return** sets the value in the input field. A popup dialog appears if the resulting value is out of range (for heaters).

**Up/Down** Increase/decrease the value of the current setpoint of one unit, and immediately effect. A popup dialog appears if the resulting value is out of range (for heaters)

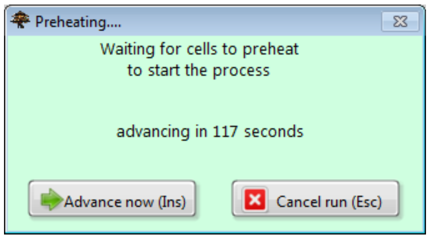
Otherwise:

**F1-2, F5-F12** toggle various buttons, as indicated in the respective labels. F1 and F2 bring the typing focus in the corresponding input fields; all the other leave the focus in the current input field if it was already in one.

**Escape** abort a running process

### 3.5.2 Process execution

#### 3.5.2.1 Starting the growth


Preheating....

Waiting for cells to preheat
to start the process

advancing in 117 seconds

In absence of any other more sophisticate scheme for controlling adequate preheating of all cells (with preparation sequences which may be special for delicate hardware), the start of a run simply initiates a countdown before the first layer. The countdown time is set in the general plant configuration (3.2.2), and the initial waiting phase can be skipped iof the operator sees that the plant is in good order to start. The process can also be aborted at this point, if something turns out to be not in order, or if the process was started erroneously.

#### 3.5.2.2 Process running

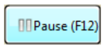
The cell status according to the schedule (idle, standby, preparation, temperature reached, ramping) is shown by the name of the cell. The background turns to fuchsia when the temperature is off of more than 1°C from the setpoint.

#### 3.5.2.3 Automatic/manual status of a control

When not executing a process, the state of these buttons is indifferent to the immediate control. During a growth process instead, any control in Auto state is driven by the requirements of the process. To override them, the control has to be toggled as Manual. The Manual state is persistent until not resetted to Auto. That is, advancement of the process won't change setpoints of any control set in Manual mode, until that is not reset to Auto.

To be clear, setpoint changes are commanded at precise points of time during the execution of the process. They are change events and not values of status. When a given control it is set back to Automatic, it won't immediately reset to the value that *it should have taken if it had been earlier Auto*; it will only change to the next process-dictated value in due time.

#### 3.5.2.4 Pause



During pause:

- all cell shutters but those of group V elements are closed. Group V elements are left open to maintain a protective atmosphere.
- timers are frozen at the point reached
- no setpoint is changed automatically, but any control can be changed like in immediate execution mode.

Other properties of the process in execution persist; for instance it is not allowed to change the plant configuration, and the growth recipe can be edited only past the current layer.

#### 3.5.2.5 Skipping steps

There are two buttons for advancing the process and skip to a later point of the schedule:

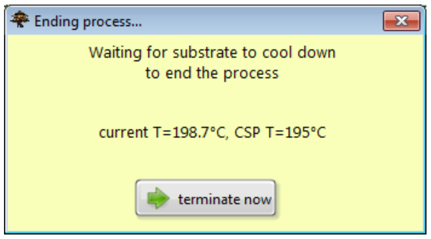
**Jump** the process jumps to the beginning of the desired layer in the growth table. If the destination is within a superlattice, the jump is implicitely to the first iteration of that layer. It is allowed to jump to a destination layer before the one currently executing, even if that may be of little practical use; that it is possible is helpful in debugging simulated processes. There is also no enforced limitation such that valid destinations may only be the first layers of a superlattice.

**Next** goes to the following process step, which may happen to be the next layer during a linear process, or the beginning of the next iteration of a superlattice, if called on the last layer of a periodic sequence.

There has been no request and thus there is no provision for jumping to an arbitrary time in the middle of a given layer, or to jump to the beginning of the previous step. I assume anything the like just doesn't make practical sense.

#### 3.5.2.6 Process termination

Both on normal process end and on abort, the substrate heater is sent to its idle temperature, and all cell shutters are closed, besides that of the element V cell(s) used in the last step executed. The plant then stands by, till the substrate has cooled down below the congruent sublimation point specified in the recipe (3.3.2). Only at that point the last shutter(s) are closed. The condition is shown in a modal dialog, which offers the option of terminating abruptly nevertheless.


Ending process...

Waiting for substrate to cool down
to end the process

current T=198.7°C, CSP T=195°C

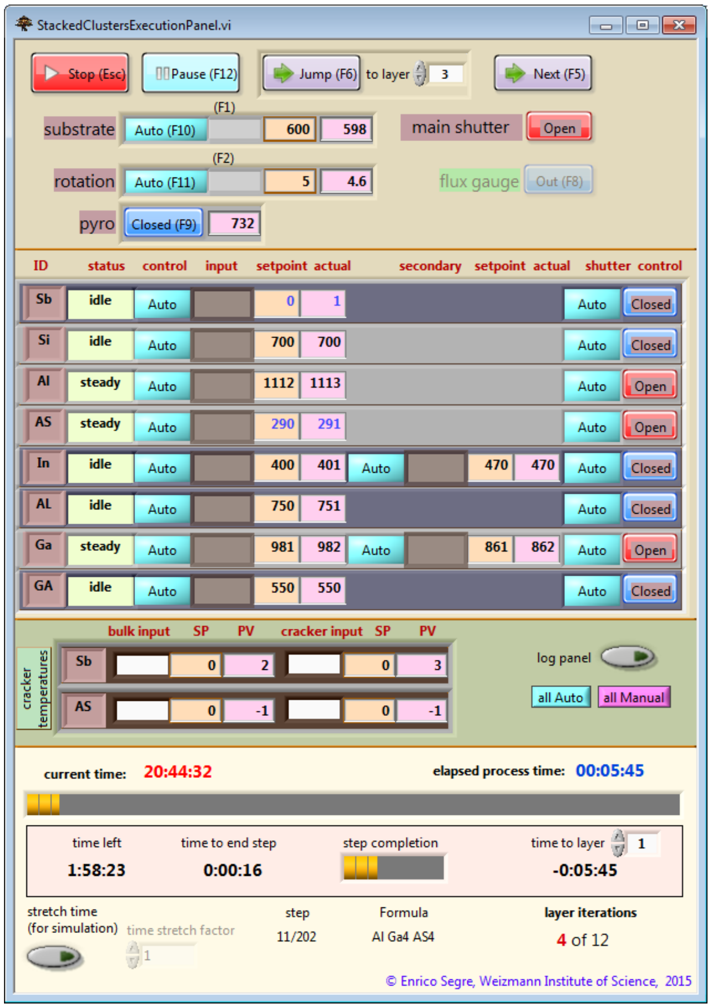
### 3.5.3 Simulated hardware

Hardware which is not identified as online at program startup is evidenced (so to say) by bleak dirty copper colored labels, like in figure; functional hardware by green labels. Grouped entities may be labeled as absent if communication attempts with one of their components failed at startup. Thus, effusion cells are marked as bad if either or all of their heaters, of the shutter, are not reached.

For development purposes it has been very useful to display fake data for absent hardware, somehow related to configured limits and process setpoints, plus random jitter.

Limitations:

- there is no option for rescanning hardware and checking if hardware temporary offline at program startup is later available, or vice versa;
- there is no indication of hardware which falls offline time after having been connected to at startup
- simulated readouts, along with real readouts, without a clear delineation of a simulation mode may be confusing.


StackedClustersExecutionPanel.vi

to layer 3

substrate (F1) Auto (F10) 600 598 — main shutter Open

rotation (F2) Auto (F11) 5 4.6 — flux gauge Out (F8)

pyro Closed (F9) 732

| ID | status | control | input | setpoint | actual | secondary | | setpoint | actual | shutter | control |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sb | idle | Auto | | 0 | 1 | | | | | Auto | Closed |
| Si | idle | Auto | | 700 | 700 | | | | | Auto | Closed |
| Al | steady | Auto | | 1112 | 1113 | | | | | Auto | Open |
| AS | steady | Auto | | 290 | 291 | | | | | Auto | Open |
| In | idle | Auto | | 400 | 401 | Auto | | 470 | 470 | Auto | Closed |
| AL | idle | Auto | | 750 | 751 | | | | | Auto | Closed |
| Ga | steady | Auto | | 981 | 982 | Auto | | 861 | 862 | Auto | Open |
| GA | idle | Auto | | 550 | 550 | | | | | Auto | Closed |

cracker temperatures

| | bulk input | SP | PV | cracker input | SP | PV |
|---|---|---|---|---|---|---|
| Sb | | 0 | 2 | | 0 | 3 |
| AS | | 0 | -1 | | 0 | -1 |

current time: 20:44:32 — elapsed process time: 00:05:45

time left 1:58:23 — time to end step 0:00:16 — step completion — time to layer 1 -0:05:45

stretch time (for simulation) — time stretch factor 1 — step 11/202 — Formula Al Ga4 AS4 — layer iterations 4 of 12

© Enrico Segre, Weizmann Institute of Science, 2015

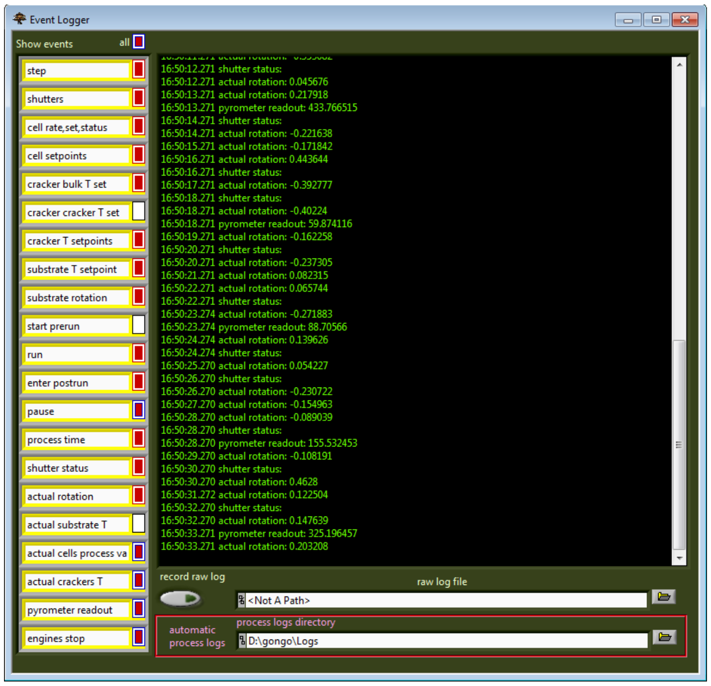
#### 3.5.3.1 stretching (actually, accelerating) process time

This is an option which is useful for debugging and quick check of the scheduling sequences. When on, it is possible to accelerate (or decelerate) the process clock by an arbitrary factor. The effect while running in simulation mode is that of accelerating all animations; it is quite meaningless and probably disrecommended to use this modality with real hardware, for two reasons:

1. large stretch factors will increase the communication rate with devices, leading to communication errors when too high;
2. not only layer heights would be proportionally decreased, also preparation times would become too short; accelerating the process time is obviously not a way to scale down sizes, if that ever had any sense.

## 3.6 Logger

The logger window is opened by the dedicated button in the **Plant control** panel. It is accessory, and besides being the place for defining once for good the process log directory (3.6.1), it is mostly used only for debugging.



| Show events | |
|---|---|
| all | ✓ |
| step | ✓ |
| shutters | ✓ |
| cell rate,set,status | ✓ |
| cell setpoints | ✓ |
| cracker bulk T set | ✓ |
| cracker cracker T set | |
| cracker T setpoints | ✓ |
| substrate T setpoint | ✓ |
| substrate rotation | ✓ |
| start prerun | |
| run | ✓ |
| enter postrun | ✓ |
| pause | ✓ |
| process time | ✓ |
| shutter status | ✓ |
| actual rotation | ✓ |
| actual substrate T | |
| actual cells process va | ✓ |
| actual crackers T | ✓ |
| pyrometer readout | ✓ |
| engines stop | ✓ |

```
16:50:12.271 shutter status:
16:50:12.271 actual rotation: 0.045676
16:50:13.271 actual rotation: 0.217918
16:50:13.271 pyrometer readout: 433.766515
16:50:14.271 shutter status:
16:50:14.271 actual rotation: -0.221638
16:50:15.271 actual rotation: -0.171842
16:50:16.271 actual rotation: 0.443644
16:50:16.271 shutter status:
16:50:17.271 actual rotation: -0.392777
16:50:18.271 shutter status:
16:50:18.271 actual rotation: -0.40224
16:50:18.271 pyrometer readout: 59.874116
16:50:19.271 actual rotation: -0.162258
16:50:20.271 shutter status:
16:50:20.271 actual rotation: -0.237305
16:50:21.271 actual rotation: 0.082315
16:50:22.271 actual rotation: 0.065744
16:50:22.271 shutter status:
16:50:23.274 actual rotation: -0.271883
16:50:23.274 pyrometer readout: 88.70566
16:50:24.274 actual rotation: 0.139626
16:50:24.274 shutter status:
16:50:25.270 actual rotation: 0.054227
16:50:26.270 shutter status:
16:50:26.270 actual rotation: -0.230722
16:50:27.270 actual rotation: -0.154963
16:50:28.270 actual rotation: -0.089039
16:50:28.270 shutter status:
16:50:28.270 pyrometer readout: 155.532453
16:50:29.270 actual rotation: -0.108191
16:50:30.270 shutter status:
16:50:30.270 actual rotation: 0.4628
16:50:31.272 actual rotation: 0.122504
16:50:32.270 shutter status:
16:50:32.270 actual rotation: 0.147639
16:50:33.271 pyrometer readout: 325.196457
16:50:33.271 actual rotation: 0.203208
```

raw log file: <Not A Path>

process logs directory: D:\gongo\Logs

The idea behind it: the whole program is compartimentalized, and the **Plant control** panel sends execution messages to a certain set of execution engines (see 5). Those in turn send back some acknowledgment and some readout messages to the GUI; the GUI interprets them and reacts according to its status. The logger panel allows to inspect the stream of these messages, filtering out only some kinds of, as checked in the column at the left of the terminal window.

### 3.6.1 Automatic process log

For that to happen, a default logs directory has to be selected once (and is remembered subsequently across sessions) in the logger panel. These logs are tab-separated text files with default name (same as the recipe file) and extension (.rec), like

```
Process No.: simple_ramp
Start time & date: 22:31:49 16/07/2015
======================================
CRACKERS
Sb bulk=101°C cracker=118°C
AS bulk=49°C cracker=62°C
======================================
    Time Layer# Sb Si Al AS In AL Ga GA subst.  pyro.  rot.
00:00:00 1 1 C 701 C 990 C 90 O 401 C 750 C 871 O 550 C 299 926 2.2
00:01:20 2 0 C 701 C 989 O 91 O 401 C 751 C 872 O 550 C 300 564 2.4
00:02:24 3 0 C 700 C 989 O 90 O 400 C 750 C 871 O 550 C 299 554 1.6
```

The content of the process log files is fixed, and independent of other settings in the Logger panel. The cracker temperatures reported are those at the beginning of the growth. Lines are added to the file as long as layers start execution; times in the first column are scheduled process times (don't reflect manual pausing if there was such). The temperatures/setpoints and shutter stati reported for each layer line are the first values read after the start of each layer.

### 3.6.2 Raw events log

Raw log files are a text dump of the events shown in the window, as filtered by the left list. Raw event log files are intended mainly to be used for debugging purposes, or for timing analysis, with external textual tools to be devised ad hoc and per need.

## 3.7 Session parameters

In addition to the hardware and system configuration is stored in the file `plantconf.cfg` in the Application Directory (3.2.3), the following settings are remembered across sessions:

- the last path used for process log files
- the last path used to load/save recipes
- the last path used to load/save configuration files

and are all stored on exit in the file `GrowthPaths.ini` in the Application Directory. [the scheme might be changed in future to support multiple plant configurations]

# Part II

# Design philosophy and implementation choices

In what follows I attempt to note down some of the principles which led to the make of the present code, for future reference. I don't aim to a complete description of the code workings; for that I hope that either the source files organization and naming, or the code itself are sort of self-explanatory. It is certainly wishful thinking, but a complete code documentation, for a codebase which may still change, would be sort of out of place.

The development of the code certainly suffered of inadequate initial formalization of its requirements, and of subsequent feature creep. Laying out the development plan was in a way a red queen race, attempting to include a priori flexibility in anticipation of possible future requests. With the risk of early overgeneralization.

The code itself has to accommodate for:

- flexibility in configuration with different hardware, interchangeable in time or between different operated plants
- abstraction of the operation from the actual hardware employed, which may sometimes be difficult to achieve (6.1)
- immediate SCADA of the hardware, with convenience shortcuts
- editing growth process recipes, which are laid out according to a number of notational conventions, and must satisfy a number of operational constraints; editing must be assisted, convenient in filling up derived data; and as much as possible validated. Recipe validation and in general error handling was (stupidly) considered an unnecessary luxury in the requirements phase, obviously a distorted attitude.
- execute process cycles in supervised mode, with reliable timing.

Chapter 4

# Dataflow and internal representations

In order to avoid a dependency hell, I introduced a multistage internal representation:

1. the cell configuration defines the available elements and the role they can take in a material formula (4.1)
2. the material formula determines the assisted editing of the layer of the growth table
3. the layer line *and its relation with its antecedent and followers* (unrolled wise) determines the sequence of hardware setpoints needed to execute a process. The layer table is translated into a sequence of process steps (4.2)
4. The process steps computed from the growth recipe are downloaded to the main execution engine, the Execution-EventEnqueuer, which first of all translates it into four different arrays of schedule elements (4.3), destined to four different hardware engines (5)
5. The ExecutionEventEnqueuer is a state machine caring for the execution status of the process, like start, stop, pause.
6. When the process is executed, the **ExecutionEventEnqueuer** polls the process timer every 5ms, and dispatches execution messages to the hardware engines. These messages are writes, if scheduled events falling in the time slot are found; or periodical reads, when time for them is ripe.
7. To mix up enough things, communications between the engines and the GUI is implemented via several message passing queues:

- the **Plant control** GUI communicates both with the **ExecutionEventEnqueuer** and with the executioners via a message passing queue, the *GUICommandQueue*; care is taken so that only one and only one among the main engine and the executioners[1] consumes each possible message.
- the **ExecutionEventEnqueuer** sends commands (scheduled events) to the four hardware engines, via four dedicated queues
- each of the five reports back to the GUI via an *EngineResponseQueue*. The GUI is the only consumer of these responses, but duplicates them on reception to a further queue consumed in turn by the logger.

## 4.1 From cell configuration to picker

The plant configuration structure is stored as a cluster of:

- an array of *EffusionCells*
- a cluster of definitions of main objects
- a cluster of timings

[1]pun intended.

### 4.1.1 Format of the plant configuration file

Early in devising what format to use to save configurations, I decided to go for “xml dump of whatever everything”. This was a working choice, especially at a stage where it was completely unclear what kind of hardware is part of the plant and how it is hierarchically related, and which parameters of it constitute a configuration. Within the program the configuration structure organized itself as an heterogeneous cluster, and a mindless xml dump was at least a solution. Not free of maintenance issues, though:

- the configuration file is all text, but for programming convenience it is written in such a complicate way that it is quite difficult to edit it sensefully in a text editor
- As of now, the project includes a `plantconf.cfg` (thus if you download a zip source it is in there), and the installers I create also copy it in the destination directory. Thus, your custom created configurations are definitely going to be overridden, unless you create a safety copy.
- It is becoming rarer lately, because the design is converging, but not impossible, that if version after version I have to alter the structure of the configuration, old configuration files won’t load in new version, or load only a part of the parameters. One such change could be, for example, adding a flag "delicate" to crackers which need a slow sleep procedure (issue #57).
- In case of old configurations incompatible with new programs, I may be able to adapt the files for the new needs, but with some effort, and best if I’m only one version behind. I really haven’t yet thought at a way to automatically maintain backward compatibility of such files.
- Because of the identification of the cell configuration with the array *CellProperties*, the configuration ends up containing some physical, contingent information as part of it, such as *PhysicallyPresent* and actual setpoint values.

Later in the development, i.e. with commit 7120745, I phased out xml configurations in favour of the new compact .cfg format.

What lead to that:

- xml configurations were an easy solution (just formatted dump of the labview structure defining the components of the plant, using a built in tool), but were overbloated, and included not only configuration parameters but also state variables (presence, setpoint).
- serialization is not trivial, because the alternatives considered (native labview Json, JSON LabVIEW, AQ Lineator) are inadequate. the native Json is extremely limited in types of data handled; json-labview does a very good job in recursively serialize (variant) data down to its components, but handles the serialization of OO objects only if their class are declared as children of a *Serializable* superclass, which involves some code changes; and more importantly, it does not provide any mechanism to deserialize the objects back to their structure without extra coding; and the Lineator does not provide a particularly compact output.

So I tediously created my own set of serialization functions for every possible component of the big PlantConfiguration cluster. Where I could, I coded a single format string good both for the **ConfigToText** and **TextToConfig** method VIs, to concentrate changes in a single place. The choice adds the overhead that whenever the need for changing the configuration structure arises, the proper methods have to be updated too.

The result is a configuration format in which each object corresponds to one line of the file, inspired by Json but much more rigid (to avoid a true parsing, it’s only a formatted string scan) and compact. The advantages are compactness, terseness and hence better readability, and absence of spurious state variables.

## 4.2 From growth table to ProcessSteps

The best labview widget to represent the growth recipe was the Table. However Labview Tables are limited in that they only support text content, have limited and fixed editing bindings, which are overridden with lots of work and side effects, and are very slowly decorated.

Everything depends on itself and its mother’s ass.

A line of the growth table includes only information about the components of that layer, and moreover some timing information whose significance depends on the composition, whereas the execution of the process needs to know exactly what to do with all components of the plant: what to do with all cells, when to change temperature setpoints, also accounting for preheats and idling. Furthermore, these cell settings may differ in relation to a layer being grown the first, a repeated or the last time within a superlattice.

For these reasons, it was convenient to introduce in the program an intermediate translation of the growth table as an array of ProcessSteps. Superlattice repetitions are first of all unrolled, so that the correspondence $\{$layer,iteration$\}\longrightarrow$step is resolved.

### 4.2.1 numeric representation of the growth table

The growth table was designed using the labview table widget, as it was the only practical way to build a tabular editing mask with some chance of moving focus programmatically, colorize and format entries programmatically, and so on. However, its content is the text which is displayed and nothing beyond. Where an entry is a computed number, truncated to few significant digits for display convenience, that could become a problem if the number is further used in computations. Truncation errors may propagate. Ways out of the situation may be:

- associate a parallel numeric array which mirrors the table at full precision; it will be a lot of work to keep it in strict sync with all the table editing actions; many editing actions will produce a floating result, which is inserted in the mirror, and reformatted back to the table.
- at the moment of compiling the table into ProcessSteps, instead of blindly reading the cell string, recompute all derived numbers using again the machinery which produced them, line per line according to the material and its rules. That may mitigate or perhaps even circumvent the problem.

### 4.2.2 the class ProcessSteps

Each ProcessStep is a cluster detailing:

- corresponding role in the Growth table: original layer, formula, time of start and end layer growth
- general scalar effects of the layer, like thickness, rotation speed, substrate temperature
- derived rate information, i.e. growth rate of the base III element, total growth rate at the end of the previous step and at the end of current step: they all enter in formula computations and operational cell settings (previous rate notably for ramping layers)
- identification (indices) of solute III cells used and component rates demanded from them
- an array detailing for every cell of the plant: whether that cell is in use (=shutter open) in the step, the required initial and final rates and control setpoints, with in range flags.
- an array, detailing, for every cell of the plant, a vector of (process) times at which rates, setpoints and cell status have to be changed. Unused cells list anyway an idle setpoint at the initial time of the layer; this avoids tedious status bookkeeping when step hopping during process execution.

The array of ProcessSteps is built from the growth table in a number of parsing steps in **GrowthTableToProcessSteps.vi**.

## 4.3 From ProcessSteps to schedules

The class GeneralSchedule has a header with schedule time, corresponding growth table row and process step, and an action field, *set* or *get*. To ~~limit expandability~~ simplify, both the hardware writes and the periodical reads are implemented as schedule events dispatched to the engines, and the action field determines the behavior.

This class has three children with specific properties relevant to each engine: the *ShutterSchedule* has a boolean array of shutter conditions, the *CellSchedule* has rate, setpoint, cell number and cell use; the same *ScalarValueSchedule* is used for substrate rotator and temperature.

# Chapter 5

# Execution engines

with the questions always in the background - why didn't I use the DSC module (because it is not available for linux, precondition for my steady development pace), whether did I reinvent the wheel with a custom set of execution engines (now what, recipe execution with all its constraints could have been handled with a standard tool?), why don't I connect with an historical database (which? that which comes with DSC, what comes nowadays? SQL? a microsoft product?).

As long as events to be logged are a subset of those returned to the GUI queue by the execution engines (see **ReportedEventsLogger.vi**), though, there may be some basis for neat development.

## 5.1 Realtime

Reliable and repeatable growths clearly require determinism in time. Nevertheless, it would seem that the acceptable jitter in the timing of shutters could even be of the order of 100ms (comparable with the action time), and that of heaters even longer. Previous anecdotal evidence (both on programmer's and user's side) reported it as achievable with difficulty, and only under draconian safeguard measures, on the demanded desktop, non RTOS.

The proper handling of the realtime requirement would require separation between the GUI and the time critical parts, with the latter running headless on a separate realtime target, and proper intercommunication between the two. The present design of the program has taken that into due account, with the creation of self-sufficient engines. However, the provision for separation is not yet complete, as the engines communicate with the GUI through queues, whose scope is the single Application instance. The future move allowing detached run of the engines on a separate target would be to transform the queued messaging into some other, probably network based, communication mechanism.

Resource separation between GUI and engines is also not complete, as initialization is done by the GUI (namely the Plant control panel at startup and when detecting configuration changes). Partially, for the reasons in 6.1; partially because (re)initialization could be implemented better.

As for on the road performance, anyway, some analysys of the log timings has shown a jitter in fact down to a few ms, so that previous concerns seem currently non-issues. Computer powers may be larger than in previous incarnations of the control system; anyhow the present program has been seen using only a few % of the available cpu power, and in addition all the engine VIs have been assigned as running at "time critical priority".

# Chapter 6

# Adding further hardware and functionality

Tip when changing the structure of the system configuration (e.g. adding new members to the global configuration, changing the structure of a member, etc.): if structural changes are made, the old `plantconf.cfg` file will be out of sync with the new configuration structure, an error will result when **GrowthPlan.vi** attempts to read it, and an empty or incomplete configuration will result. In order to preserve as much as possible of the existing configuration, a workable way to develop structural changes is the following:

- run **SystemConfigurationPanel.vi** directly;
- load the existing `plantconf.cfg`; this reads in values for all controls;
- abort the VI with Ctrl-.
- code the required changes;
- as soon as **SystemConfigurationPanel.vi** is runnable again, run it and save a new, format-augmented `plantconf.cfg`.

To this extent **SystemConfigurationPanel.vi** needs not to be modal, or it would remain modal also aborted. OTOH, on the road the user should be prevented to do anything other than configuring while the panel is open. That is the rationale for 06ac4ef.

## 6.1 Issue: Double (triple) affiliation for devices

The design pattern suffers from the following:

- some hardware devices are compound devices, e.g. cells consist of heaters, a shutter, a valve; the main chamber–it depends how it is seen, if all is "manipulator" or further subdivided in Pyro, Substrate,...
- component devices are physically grouped in a way that crosses the compound device boundaries: all Eurotherms are on the same 485 line (and additionally analog readouts on Eurotherms), all shutters (or groups of) are handled by the same controller, together with other movimentation.
- execution engines are responsible for yet another cut of devices (e.g. all cell temperatures and setpoints, substrate temperature and pyro readout but not pyro shutter or rotator, all shutters together)

The problem with this organization is that competences are confuse. For instance initialization and deallocation of resources cannot take part in engines, because the cut of the engines is different than the hardware grouping; resource protection happens via globals and semaphoring, not via encapsulation; adding new kind of devices may turn into a jigsaw effort.

## 6.2 Case study: adding a new valve controller

In a later stage, porting the application to Compact 21, it turned out that we weren't able to drive the Veeco SMC-III controller via the serial BISYNC protocol like the older Veeco SMC on MBE, and we moved to the option of modbus RTU. The augmentation is an exemplar case, because it is self contained. We have only one class Valve, one attributes of which was the *ValveController* typedef. Here the distinct code driving different hardware is not implemented as one-per-kind subclasses, but simply as conditional code in each of the method VIs for the general class Valve, selected by the *ValveController* attribute. The work to be done hence amounted to:

- add a new item in the ValveController typedef
- go through the relevant method, **InitializeValve**, and three accessors implementing real communication with hardware, that is **ReadValvePosition**, **ReadValveTarget**, **WriteValveTarget**, adding the code for the new case

As the Valve class definition itself didn't change, there was no need to propagate the changes to the (only) configuration panel accessing it, the Cracker panel. However, the change in typedef enumeration caused a mismatch in the existing configuration files, which needed manual revision to reinstate the correct settings. As the configuration panels persisted across the change, that was easily done in the GUI.

Chapter 7

# Multiple configurations

## 7.1 Configuration and session files

My rationale for having two different files `plantconf`.cfg and `GrowtPaths`.ini is that one is the proper plant configuration, and this should be changed only by the responsible administrator, while session settings like the last directory chosen can well depend on each user. I don't know if it reflects the present way of work, but in principle many users could have their own accounts on a plant control computer, whereas only one authorized administrator is allowed to tamper with sensible hardware configuration like the number of cells or the addresses of the devices.

Codewise, at some stage I also had reasons for storing the two sets of data in two different structures and handling them in different parts of the program, but now that is blurred.

## 7.2 Multiple plants on a single computer

Now whichever scheme has to work on three computers as of now, where one of them needs to have an easy way of accessing two different configurations. The doubled application directory was a possible solution, but I could envisage also others: a single application with a prominent drop-down menu for chosing the plant (how do we know what are the options, and change them? Perhaps at startup some directory "Possible Configurations" is scanned. What would be the default, etc.) (I envisage that you might even want to store historical configurations for a given plant, i.e. say be able to open an old recipe with the configuration the plant had years ago, in order to check what was going on).

Another solution could be keeping a single application directory, but copying twice the executable with different names; use as default configuration not `plantconf.cfg` but `NameOfTheExecutable.cfg` or something the like; revert that `GrowthPaths.ini` is written to the User Application directory, also with a name composed with the name of the program. This way we could both handle different permission levels, different settings for different users, and different plants, with the slight waste of having two copies of the executable and having to do that manually after the installation. The hindrance is that also the file `MBEGrowthController.ini` containing the default font sizes has to be renamed accordingly. I could write a post-installation script for that; there is no end.

Another possible option is to set up things so that clicking on the (shortcut to the) configuration file, the executable (a single copy of) opens loading that file. I'd have to study how to do it smoothly in Labview.

# Chapter 8

# Source control and build

## 8.1 Git repository

Development has been source-tracked all the way through. One copy of the source repository is at the WIS-internal project repository, accessible only within WIS firewall. It is a private project, so sign in on the WIS gitlab server for access first, and then ask Enrico to be named as member, in order to see it.

LabVIEW source files are stored with compiled code removed, to prevent useless binary divergence. Some instrument manuals, some requirements documents and the sources of this manual are also stored in the same repository.

## 8.2 Issue tracker

WIS-internal issue tracker. As part of the above, only accepted members, with minimal status of Reporter can write tickets.

## 8.3 Building and distributing the application

There is a technical issue about building the executable application with the labview compiler, documented in issue #56. Labview crashes apparently because of the way chosen to display the software revision and the build date, i.e. getting on the main window the information from an ancillary subVI which is programmatically filled with that data and saved as prebuild step. The crash seem to be done to the collision of former and new version of this VI in the compiled objects cache, and the action required is to clear it after unloading the project from memory. To partially overcome that loophole, a build script external to the project has been created, which also cares for additional steps like renaming the final installer reflecting the git revision id. Such steps themselves wouldn't be possible with the facilities of the project build. Clearing the object cache though may be still needed before running that script.

# Chapter 9

# Alternative ideas

First, I'm not at all convinced that functionally equivalent software doesn't exist, either commercially or as, less likely, open sourced. In fact I'm aware of one open source example in VB, Faebian Bastiman's (seen april 2015). Commercial products include Veeco's Molly, Riber's Crystal.

Argument could be on convenience of use, or suitability for the purpose, for the in-house conventions and habits. I have uderstood that here the orientation is for the execution of many one-of-a-kind processes, with the option of on-the-fly adjustments, hence the stress on the convenience of input of the recipe, rather than infinite repetition of an industrial process, for which it is affordable that the recipe is programmed only once in a while, even in a rigid and less convenient form.

I speculate that the labview infrastructure put together to control the hardware could be augmented by a scripting modality of control.