A way to realise

The PVE Lab

© The PVEout project within the European 5th framework, NRU

1 INTRODUCTION **3**

2 TIMETABLE **9**

3 THE PVE LAB **4**

3.1 MAINGUI 4

3.1.1 Proposed menus *5*

3.1.2 Proposed task *5*

3.1.3 Proposed Progress window *6*

3.1.4 Proposed information bar *6*

3.2 BEHIND PVE LAB 6

3.2.1 Main- and localGUI *6*

3.2.2 CheckIn, Checkout and the registry file *7*

3.2.3 Macro file *7*

3.2.4 Log file *7*

3.2.5 Common functions for the PVE Lab *7*

4 INITIATED SYNTAX FOR THE PVE LAB **9**

4.1 MAINGUI 10

4.2 WRAPPER 11

4.3 FUNCTIONS CALLED BY A WRAPPER 13

4.3.1 varargout=checkIn(METHOD,varargin) *13*

4.3.2 varargout=checkout(METHOD,varargin) *15*

4.3.3 hh_localGUI=name_localGUI(varargin) *16*

4.3.4 varargout=name_method(varargin) *17*

4.4 REGISTRY FILE AS A TEXT FILE 18

4.5 THE LOG FILE AS A TEXT FILE 21

5 PROGRAMMING GUIDELINES **21**

6 VOCABULARY **21**

© The PVEout project within the European 5th framework, NRU

1 Introduction

One success criteria for the PVEout (Partial Volume Effect) project within the European 5th framework is to make a common interface for a product that can do a full partial volume correction of a PET/SPECT scan using a high resolution MR scan.
At the Firenze meeting, Italy, mid November 2002, the partners agreed to make a proposal on how the different methods in the PVE process could be integrated through a user-friendly graphic user interface (GUI). Further, the partners agreed in using the Matlab program from MathWorks Inc. for implementing the GUI.
A major advantage using Matlab as platform is the non-dependency of the operating system (Linux or Windows), the worldwide use of Matlab within medical community, and an easy way to realize the GUI.

The Copenhagen group, NRU, would make the preliminary description. The result is given in the following document where a full (preliminary) description on how an user friendly interface could be realized, code be written and, different methods integrated. Further a proposed prototype of a GUI is created in Matlab.

This document contain the following sections:

In section 2 an introduction is given on how the proposed GUI of the PVE Lab can look like. Further there is a description on how a method is thought to be implemented in the PVE Lab. A method is a program from one of the partners doing e.g. segmentation, registration or like.

Section 3 is a timetable which gives a guess of what to do and time to spend at the different blocks in the PVE Lab. Here the meeting in Caen, January 2003, is important.

Then in section 4 is given an initiated syntax description of the different common functions in the PVE Lab.

Welcome to the idea of:

The PVE Lab

© The PVEout project within the European 5th framework, NRU

2 The PVE Lab

The PVE Lab consist of a main GUI that offer all the methods to do a PVE correction of a low resolution PET or SPECT scan (functional image) given a high resolution MR scan (structural image).

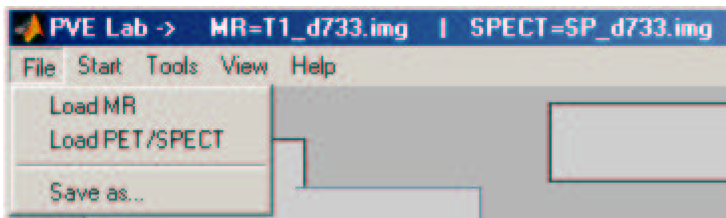
2.1 MainGUI

In Figure 1 is shown a proposed prototype of the user-friendly GUI made in Matlab. The GUI is referred as the *mainGUI.* In section 2.1.1, 2.1.2, 2.1.3 and 2.1.4 a more detailed explanation is found.

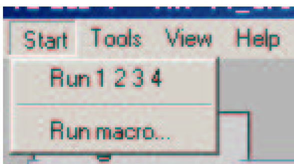
Figure 1 Proposed prototype for a main GUI for the PVE Lab where the user-friendly interface is a basic requirement. The main GUI consists of: *Progress window,* where the different methods can visualize results. *Tasks* are shown with numbers (1,2,3 and 4) as Registration, Segmentation, Atlas and PVEcorrection. Different *Methods* can be selected in each task as the case for the registration task. *Play button* activate actual task and *glasses button* show the result in the progress window. *Information bar* is where information can be given to the user. The *Menu* consists of File, Start, Tool, View and Help.

© The PVEout project within the European 5th framework, NRU

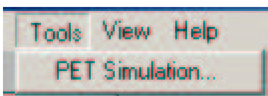
2.1.1 Proposed menus



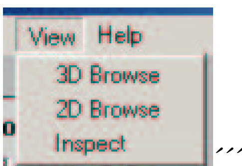
File menu: Load and save possibilities, more can be added. Name of loaded MR and SPECT/PET scan are shown at the top of the widow.



Start menu: Run a sequence of all tasks (registration, segmentation etc.) with chosen methods or load and run a recorded macro.

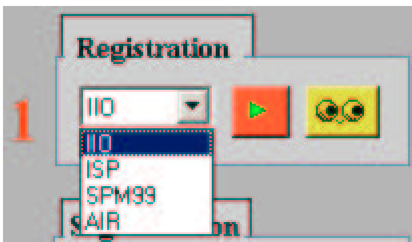


Tool menu: Chose a PET simulation instead of a real PET scan. Other additional tools can be added.



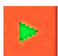
View menu: Different methods can be used for viewing the result of MR, PET or SPECT scan after a task or PVE correction.

2.1.2 Proposed task



Task and Methods: A *task* is e.g. registration, segmentation Atlas and PVEcorrection. A PVE correction consists therefore of a sequence of tasks. In each task a *method* can be chosen between one or more available methods. In the registration task there e.g. exist four different methods: IIO, ISP, SPM99 or AIR. In this case the IIO method is chosen.

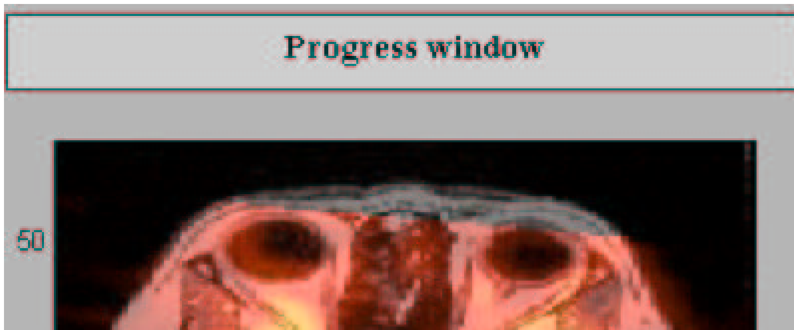
View result: Opportunity to view result from given task. The result can be viewed in the progress window.



Run Task: Opportunity to run only a given task with chosen method.

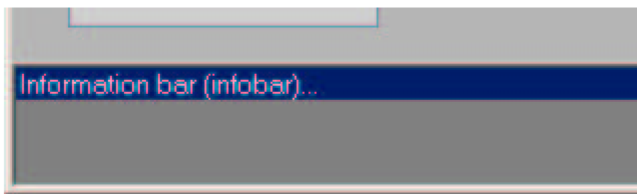
© The PVEout project within the European 5th framework, NRU

2.1.3 Proposed Progress window



Progress window: A method can show results in the main GUI 'on the fly' or when finished. Both plots and images can be shown in the progress window. A handle will be given to the started method.

2.1.4 Proposed information bar



Information bar: User or log information can be received in the information bar. The infobar can also be used as an online help menu.

2.2 Behind PVE Lab

The success and stability of a program, as the PVE Lab, is both defined by the mainGUI as the proposed prototype in Figure 1 and the structure of the code behind the mainGUI, shown in Figure 2. The basic idea of the code structure behind the PVE Lab is to split apart the *GUI code* and *working code*. The GUI has to interface to the user and give a good overview of the PVE Lab, its possibilities and the state in a PVE correction process. The working code is for specialists and includes code from methods doing the segmentation, reslicing etc. Further, code that handle the set-up of the methods and controlling the sequence of a PVE correction process will be working code.

2.2.1 Main- and localGUI

In the PVE Lab there exist two types of GUI: The mainGUI and the localGUI.
The **mainGUI** represent the PVE Lab (Figure 1) and reacts on an *action,* such as a pressed button or selection of a menu. When an action is registered a matched callback function is called. This is seen as the *Actionname_callback* in the flow diagram, Figure 2. If exist the *Actionname_callback* call a wrapper that match the chosen method, see Figure 2. The idea of a wrapper is to shield a method from other methods and GUIs. This should make it easier to overview the code because the functions is expected to have a limited size and force the programmer to write structural and general code. In this way it should also be easier to track down failures/problems and adding new methods. Inside each wrapper code written for each method is called.

The **localGUI** is designed for a given method and therefore only exist within the wrapper, as shown in Figure 2. The localGUI has the possibility to send plots or images to the progress window and text to the infobar in the mainGUI. Further it can act as a configurator with its own user interface, or as a two-way communication link between the user and a manual method as when registration two images. It should be noted that whether a localGUI is created is a decision for the programmer of the method.

2.2.2 CheckIn, Checkout and the registry file

Working code is code without any GUI as the case for methods doing e.g. the segmentation or the reslicing. The methods are (typically) programmed by the designer of the given algorithm and is placed within a wrapper, as shown in Figure 2.
The heart of the PVE Lab is the two working code functions *checkIn* and *checkOut*, see Figure 2.
The checkIn function is controlling the parameters for a given method before it is called e.g. if files are available. Further it checks whether a method must be executed at the given state in a PVE correction process or not. If not the wrapper is forced to return to the mainGUI. The checkIn function reads a *registry* file to get information upon the actual state of the PVE correcting process. The stored information in the registry file could be chosen methods, path, input output files and etc. Further the registry file keeps information needed for a method to be initialised. The registry file could be a text file, then it always can be read and edited with a normal text-viewer.

The checkOut function is called when a method has finished its work, see Figure 2. The function updates the registry file so the next state in the PVE correcting sequence can be executed. Further it should cleanup used memory to avoid memory leak, and if the localGUI is used it control that it is shutdown correctly.

2.2.3 Macro file

A macro file gives the possibility to repeat a recorded PVE correction sequence. The macro file is recorded by saving the functions and arguments of activated callback functions in a PVE correcting sequence. It should be possible for the user to load and execute a recorded macro file from one of the menus in the mainGUI see section 2.1.1.
The macro file should be a text file, then it always can be read and executed by Matlab.

2.2.4 Log file

The log file saves all necessary information of functions call that can be used to track down failures or see activated functions. The log is meant to be placed within the wrappers.
It would be preferred that the macro file is a text file, then it always can be read and edited with a normal text-viewer.

2.2.5 Common functions for the PVE Lab

To generalise the code both the checkIn and checkOut functions are common for all wrappers. This include functions reading and writing to the Log, Registry and macro file.

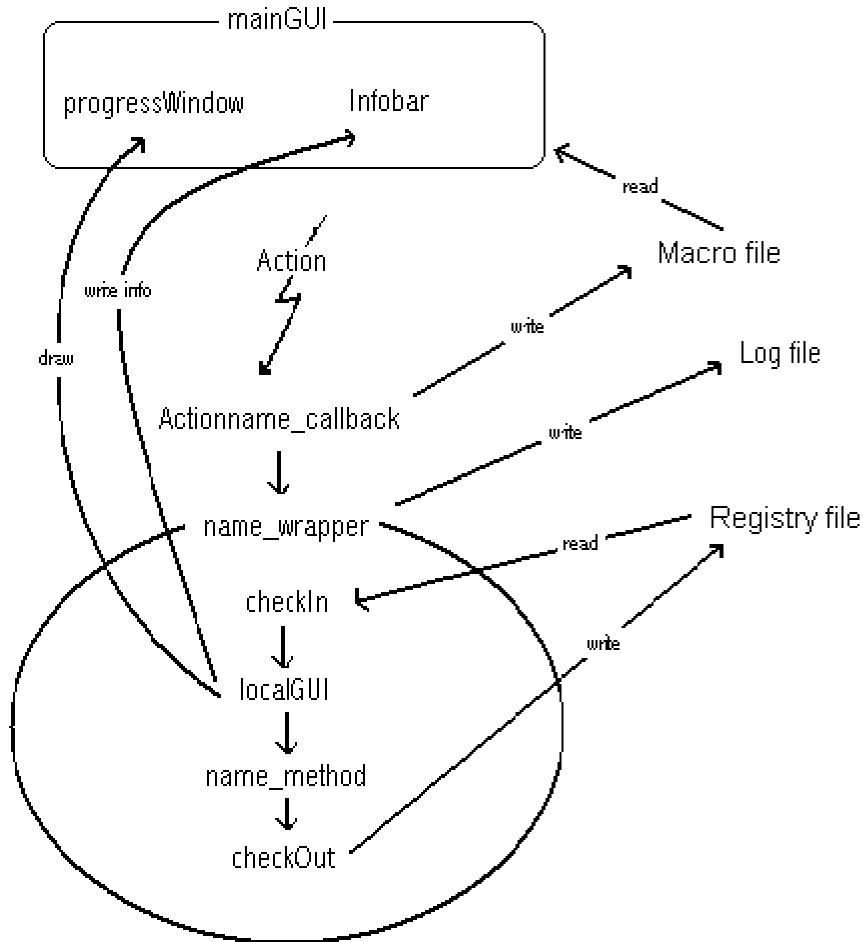


Figure 2 Basic flow diagram of the PVE Lab. When activation (e.g. button pushed) is detected in the mainGUI a given callback function and then a given wrapper is called. The wrapper shields the following function from the mainGUI except a handle to the progress window where to draw graph or images, and a handle to the Infobar where to write information. Extern files are Log file where log information are stored, a registry file that holds information on the status of the actual PVE correcting process and a macro file so a PVE correction sequence can be repeated.

3 Timetable

This timetable is only a rough guess of time to spend at the different blocks in the PVE Lab. It is thought that at the proposed meeting in Caen january 2003 the partners will agree in a finalised timetable of the PVE Lab and the blocks containing it.

Caen January, 19.-20.th 2003:

Feedback: New ideas, define and split the project into blocks. Discuss milestones on each block.

Preliminary, what to do and app. time to spend:

MainGUI: (1.5 month)

Make a design of the PVE Lab (a proposed is given), define names of variables and realise it.

CheckIn and checkOut: (app. 2.5 months or more)

Two general functions -> Very important to be well defined. Which input/output arguments should be used. How should the registry file be designed so it can full fill the demand from the two functions. The design is very depended upon the input/output specification of the different methods by the individual partner.

LocalGUI: (0-1.5 months)

GUI for the individual methods. How did the partner intent the GUI to look like and what should be possible in the localGUI: Configuration, only information or a two-way'ed communication between the localGUI and the method?

Individual methods from the partners: (0.5 month)

Detailed description of input and output arguments.

© The PVEout project within the European 5th framework, NRU

4 Initiated syntax for the PVE Lab

4.1 MainGUI

Description:

The mainGUI does a callback when activation is registered in the PVE Lab. Activation is defined as pushing a button, selecting a menu or like. Each activation is connected to a callback function which does nothing but pass arguments to a specific wrapper.

Syntax: (standard in Matlab ver 6.1 and later)

```
name_callback(h, eventdata, handles, varargin)
```

NOTE: The '*name*' is user defined, but '*_callback*' is Matlab defined.

Arguments:

- **`h:`** Callback for object's handle (structure).
- **`eventdata:`** Reserved for future use by Matlab.
- **`handles:`** Structure containing handles of all components in the mainGUI.
- **`varargin = [arg1,arg2,..,argN]`**:
  `arg1,arg2,..,argN`: Future or user defined input arguments

Calling:

A specific wrapper

External call:

Pseudo-code:

```
Call a wrapper
Return to mainGUI
```

© The PVEout project within the European 5th framework, NRU

4.2 Wrapper

Description:

A wrapper is a function that interfaces between the mainGUI (through a callback) and a given method. The wrapper takes care of the initialisation of the method, starting up the localGUI if needed and afterward cleaning up before returning to the mainGUI.

Syntax:

varargout=name_wrapper(varargin)

name_wrapper: '*name*' is user defined but should be closely related to the name of the following task. '*_wrapper*' is predefined.

Arguments:

- **varargin = [hh_handles,command,arg1,arg2,..,argN]**:
  hh_handles: Structure containing handles of all components in the mainGUI.

  command: (more argumets can be added…)
  - **check_Status**: Via the registry file is checked if the situation is correct
  - **run_Method**: Run given method
  - **run_MethodShowResult**: Run given method and show result
  - **kill_LocalGUI**: Kill localGUI after a good property
  - **no_LocalGUI**: Run method without localGUI use default parameters
  - **all_LocalGUI**: Let the progress window use all the mainGUI

  arg1,arg2,..,argN: Future input arguments

- **varargout =[status,arg1,..,argN]**:
  status: (more states can be added…)
  - **Run:** Method is checked-in
  - **err_Registry:** Can't find registry file
  - **err_infile:** Can't find input file
  - **err_outfile:** Can't find output file
  - **err_method:** Can't find program
  - **err_state:** Wrong state e.g. segmentation before registration

  localGUI_active: GUI is active

  arg1,arg2,..,argN: Future input arguments

Calling:

- `varargout=checkIn(hh_handles,varargin)`
- `hh_localGUI=name_localGUI(varargin)`
- `varargout=name_method(hh_localGUI,varargin)`
- `varargout=checkOut(METHOD,varargin)`
- `An other wrapper(METHOD,varargin)`

Extern call:

Pseudo-code :

Check-in given method and return if check-in is not possible.
If needed, start up localGUI for given method
Call and initialise method and run it

© The PVEout project within the European 5th framework, NRU

If exist close localGUI
Check-out the method by updating the registry file
Cleanup before return to mainGUI
If wanted call an other wrapper

© The PVEout project within the European 5th framework, NRU

4.3 Functions called by a wrapper

Below is a short description of the functions used by the wrapper, what they are thought as in- and out put arguments and function.

- `varargout=checkIn(METHOD,varargin)`
- `hh_localGUI=name_localGUI(varargin)`
- `varargout=name_method(hh_localGUI,varargin)`
- `varargout=checkOut(METHOD,varargin)`
- `An other wrapper(METHOD,varargin)`

4.3.1 varargout=checkIn(METHOD,varargin)

Description:

A wrapper is checked-in if actual state in the PVE correcting process is valid and needed files exist. This is known through a predefined order of states. The state of a process is stored in the registry file. If an error is detected a wrapper will not be able to checked-in and may return to the mainGUI and an error message is send to the user.

Syntax:

`varargout=checkIn(METHOD,varargin)`

Arguments:

- **`METHOD`**: Label (constant) of the method checking in

- **`varargin`**=**`[hh_handles,command,arg1,..,argN]`**

  `hh_handles:` Structure containing handles of all components in the mainGUI.

  `command:` (more argumets can be added…)

  `check_Status`: Via registry file the situation is checked for given method.
  `run_Method`: Run given method
  `run_MethodShowResult`: Run given method and show result
  `kill_LocalGUI`: Kill localGUI
  `no_LocalGUI`: Run method without localGUI use default parameters

  `arg1,arg2,..,argN`: Future input arguments

- **`Varargout`**=[**`status,MRin,MRout,PETin,PETout,h_mainfig,`**
  `h_infobar,arg1,..,argN`]

  `status:` (more states can be added…)

  `Run:` Method is checked-in
  `err_Registry:` Can't find registry file
  `err_infile:` Can't find input file
  `err_outfile:` Can't find output file
  `err_method:` Can't find program
  `err_state:` Wrong state e.g. segmentation before registration

  `MRin:` `Filename with path to input MR`
  `MRout:` `Filename with path to output MR`
  `PETin:` `Filename with path to input PET/SPECT`

© The PVEout project within the European 5th framework, NRU

PETout: Filename with path to output PET/SPECT
h_progressWin: Handle to the progress window in the mainGUI
h_infobar: Handle to information bar in the mainGUI
arg1,arg2,..,argN: Future input arguments

Calling:

External call:

Registry file: Text (or other) file with information of path and status of a PVE correcting process.
Log file: Text (or other) file with log information of the PVE correcting process.

Pseudo-code:

Check-in given method
Return with status if check-in succeeds.
If not succeed return an error

© The PVEout project within the European 5th framework, NRU

4.3.2 varargout=checkout(METHOD,varargin)

Description:

Before a wrapper can return to the mainGUI or calling an other wrapper it must be checked-out. The function updates the registry file with information about the executed method and the state in the PVE correcting process. If an error is detected the registry file is not updated before returning to the mainGUI further an error message is send to the user.

Syntax:

`varargout=checkOut(METHOD,varargin)`

Arguments:

- **`METHOD`**: Label (constant) of the method checking in
- **`Varargin`**=[**`command,MRin,MRout,PETin,PETout,h_progressWindow,h_infobar,arg1,..,argN`**]
  - **`status:`** (more states can be added…)
    - **`Run:`** Method is checked-in
    - **`err_Registry:`** Can't find registry file
    - **`err_infile:`** Can't find input file
    - **`err_outfile:`** Can't find output file
    - **`err_method:`** Can't find program
    - **`err_state:`** Wrong state e.g. segmentation before registration
  - **`MRin:`** Filename with path to input MR
  - **`MRout:`** Filename with path to output MR
  - **`PETin:`** Filename with path to input PET/SPECT
  - **`PETout:`** Filename with path to output PET/SPECT
  - **`h_progressWindow:`** Handle to the progress window in the mainGUI
  - **`h_infobar:`** Handle to information bar at the mainGUI
  - **`arg1,arg2,..,argN`**: Future input arguments
- **`VarargOut=[arg1,..,argN`**]
  - **`arg1,arg2,..,argN`**: Future input arguments

Calling:

External call:

Registry file: Text (or like) file with information of path and status of a PVE correcting process.

Log file: Text (or like) file with log information of the PVE correcting process.

Pseudo-code :

Check-out given method.

If no error update the registry file

If localGUI is used, restore mainGUI and kill/shout down the localGUI

4.3.3 hh_localGUI=name_localGUI(varargin)

Description:

Interaction with a user can be done by a localGUI. The localGUI can be used to inform about a process of a given task (e.g. segmentation), or it can be used as a 2-way communication in the case of a semi-automatic or a manual method (e.g. registration).
A given localGUI is typical designed/programmed by the designer of the given method.

Syntax:

`hh_localGUI=name_localGUI(varargin)`

Arguments:

- **`varargin=[command,h_mainfig,h_infobar, arg1,..,argN]`**
  - **`command:`** (more states can be added…)
    - **`check_Status`**: Via the registry file the situation is checked for given method.
    - **`run_Method`**: Run given method
    - **`run_MethodShowResult`**: Run given method and show result
    - **`kill_LocalGUI`**: Kill localGUI
    - **`no_LocalGUI`**: Run method without localGUI use default parameters
  - **`h_mainfig:`** Handle to mainfig in the mainGUI
  - **`h_infobar:`** Handle to information bar at the mainGUI
  - **`arg1,arg2,..,argN`**: User defined input arguments
- **`hh_localGUI`**: A handle to a structure of handles of available buttons, edit boxes, List box etc in the localGUI.

Calling:

External call:

Pseudo-code :

Set up a GUI interface, which fits a given metod.
Return handles of the localGUI

4.3.4 varargout=name_method(varargin)

Description:

A method is a program performing e.g. segmentation; registration or loading and converting between different file formats. The given method can be any executable, Matlab routine or MEX function with given input and output arguments.

Requirements:

1) MR -, PET- and SPECT shall be read and written in the analyze format.
2) Run on **both** the Linux and the Windows platform.

Syntax:

`varargout=name_method(varargin)`

Arguments:

- **`varargin=[hh_localGUI,h_Infobar,arg1,arg2,..,argN]`**
  `hh_localGUI:` Handles in localGUI
  `h_infobar:` Handle to information bar at the mainGUI
  `arg1,arg2,..,argN`: User defined input arguments (in-/out filenames)
- **`varargout=[arg1,arg2,..,argN]`**
  `arg1,arg2,..,argN`: User defined input arguments (in-/out filenames)

Calling:

User defined!!

Pseudo-code:

Defined by the designer of the actual method how to solve a given method.

© The PVEout project within the European 5th framework, NRU

4.4 *Registry file as a text file*

Description:

The registry file is a text file and contains status on finished methods in the PVE correction process and is read/written by wrappers, see Figure 2.
The name of the registry file is connected to the structural file (MR scan) with the extension *.reg.
The name of the registry file is written into the header of the analyze file format for the given structural file. This gives the possibility to continue a process later on or to track information upon which methods, structural and functional files are used in a given process.

Syntax:

<work directory><separator><argument><comment><CR>
<result directory><separator><argument><comment><CR>
<task_order><separator>< task _1><space> … <space>< task _N><CR>

(For *each* task:)

<method><CR>
<info_ task ><separator><argument><space><argument><space>...<comment><CR>
<info_ task ><separator><argument><space><argument><space>...<comment><CR>
…
<info_ task ><separator><argument><space><argument><space>...<comment><CR>

Arguments:

<separator>
‘:’ Separates commands and arguments

<work directory> (command, not case sensitive)
‘*work_dir*’ Directory containing input files can be either in the Linux- (‘/’) or the Windows (‘c:\..’) way.

<result directory>(command, not case sensitive)
‘*result_dir*’ Directory containing result files generated under the PVE correction. Can be in either in the Linux- (‘/’) or the Windows (‘c:\..’) syntax.

<task_order> (command, not case sensitive.)
‘*task_order*’ Expected order of/and tasks in registry file.

NOTE: It is possible that not listed tasks exist. Such a task or task are called *part task(s)*. In the converting task where file are loaded (below example) a two part task are used: one that load and convert a functional file and one for the structural file. In the registration task a part task do the reslicing is used.

<tak_1-N> (command, not case sensitive)
’*name_task’* name of actual task followed by ‘*_task*’.

<info_task > (command, not case sensitive)
‘*input file*’ zero or more strings for input file (no path)
‘*output file*’ zero or more strings for input file (no path)
‘*method*’ string for actual method (program) used

‘*varargin*’ zero or more input arguments for actual task
‘*varargout*‘ zero or more output arguments for actual task
‘*start time*’ mm.dd.ss start time for task (one argument)
‘*end time*’ mm.dd.ss end time for task (one argument)
‘*date*’ dd.mm.yy date of process (one argument)
‘*log file*’ string for log file
‘*user_defined_1*‘ zero or more different user defined user defined strings
…
‘*user_defined_n*‘ zero or more different user defined user defined strings

<argument> (case sensitive)
String (no spaces)

<space>
one or more spaces

<comment> = <%><string> (Not case sensitive.)
‘*string*’ User defined text

<CR>
New line

© The PVEout project within the European 5th framework, NRU

Example:

In a registration process of a MR and PET scan the following task and order has to be executed:

```
%%%%%%% registry file start %%%%%%%%%%

%header information
Work_dir: /depot/scans/ %where input files are found
result_dir: /depot/scans/resultA/ %where to place output files

Task_order: converting registration reslicing segmentation atlas
(simulation) PVEcorrection

%information on which tasks are done and their arguments
converting_task %This is a main task that's use
                  %loadTarget_taks and loadSource_task

loadTarget_task % part task Loading MR files
input file: MRname.sfh % Loadede in simple format
output file: c_MRname.hdr % Indicate converting
method: Convertsf2Analyze % Program used to solve the task
varargin:
varargout:
start time: 02.13.00 %hh.mm.ss
end time: 02.15.00 %hh.mm.ss
date: 18.11.02 %dd.mm.yy
log file: MRname.log %Where to log callbacks

loadSource_taks % part task loading PET/SPECT files
input file: PETname.hdr %
output file:
method: loadRES %program used to solve the task
varargin:
varargout:
start time: 02.13.00 %hh.mm.ss
end time: 02.15.00 %hh.mm.ss
date: 18.11.02 %dd.mm.yy
log file: MRname.log %Where to log callbacks

registration_task %This is a main task that's use
                    %reslicing_task
input file: c_MRname.hdr c_PETName.hdr
output file:
method: IIO %name of method
varargin: c_MRname.hdr c_PETname.hdr %list of arguments
varargout: A %The rigid body transformation
start time: 04.02.22    %hh.mm.ss
end time: 04.24.22 %hh.mm.ss
date: 18.11.02 %dd.mm.yy
log file: MRname.log %Where to log callbacks
other: c_MRname.air %Info of registration matrix/info saved in AIR-
format

reslicing_taks % part task Reslicing
input file: c_PETname.hdr
output file: rc_PETname.hdr
method: interpol3 %name of selected task
varargin: c_PETname.hdr A %list of arguments
varargout: rc_PETname.hdr %list of arguments
start time: 04.02.22    %hh.mm.ss
end time: 04.24.22 %hh.mm.ss
date: 18.11.02 %dd.mm.yy
log file: MRname.log %Where to log callbacks

%%%%%%% registry file end %%%%%%%%%%
```

4.5 The Log file as a text file

The log file is a text file and contain logged information of activation in the PVE Lab by copying the callbacks into the log file. The log file can be used both as a guideline for new users, to watch problems in the process line or as a Macro if the same process has to be repeated.

The name of a log file is the same as the MR file name added with the extension **.log*.

Syntax:

<callback no. *1*><CR>
<callback no. *2*><CR>
…
<callback no. *n*><CR>

!!!!!!!!!!!!!!!! NOT FINISHED !!!!!!!!!!!!!!!!!!!!!!!

5 Programming guidelines

- In the same function do not mix GUI code and other code!
- Think of the memory been used. Clean up after use!
- Keep things simple this often makes speedy methods!
- Avoid global variables!
- Write the code, as it should be checked/used by others!
- Comments and introduction in a function is a must!
- Writing handles:
  h_name: a handle to a structure of variables (pointer)
  hh_name: a handle to a structure of handles (pointer to pointer)

6 vocabulary

Task
: Segmentation, registration etc.

Method
: A program that's realize a given task, e.g. different methods to do a registration

LocalGUI
: GUI for a given method and is should be designed by the designer behind the method

MainGUI
: GUI for the PVElab

Wrapper
: Function that interface between the mainGUI a given method.

Matlab
: Platform of the PVElab, is independed of operation system

Varargin/outs
: Input and out variables between functions. The number is specific for the individual function

Callback
  Matlab do a callback when activation is registered e.g. push button.

Progress window
  Where a task can display process information (graphs or images) in the mainGUI.

Infobar
  Where information of a method can be given to the user

handle
  A ‘pointer’ to a given structure containing variables or handles. Here additional information can be stored and changed after calling and before returning of callbacks.

Overloaded functions:

Functional file
  Low resolution PET/SPECT scan in the analyze format

Structural file
  High resolution MR scan in the analyze format consist of an header file (*.hdr) and image file (*.img)

© The PVEout project within the European 5th framework, NRU