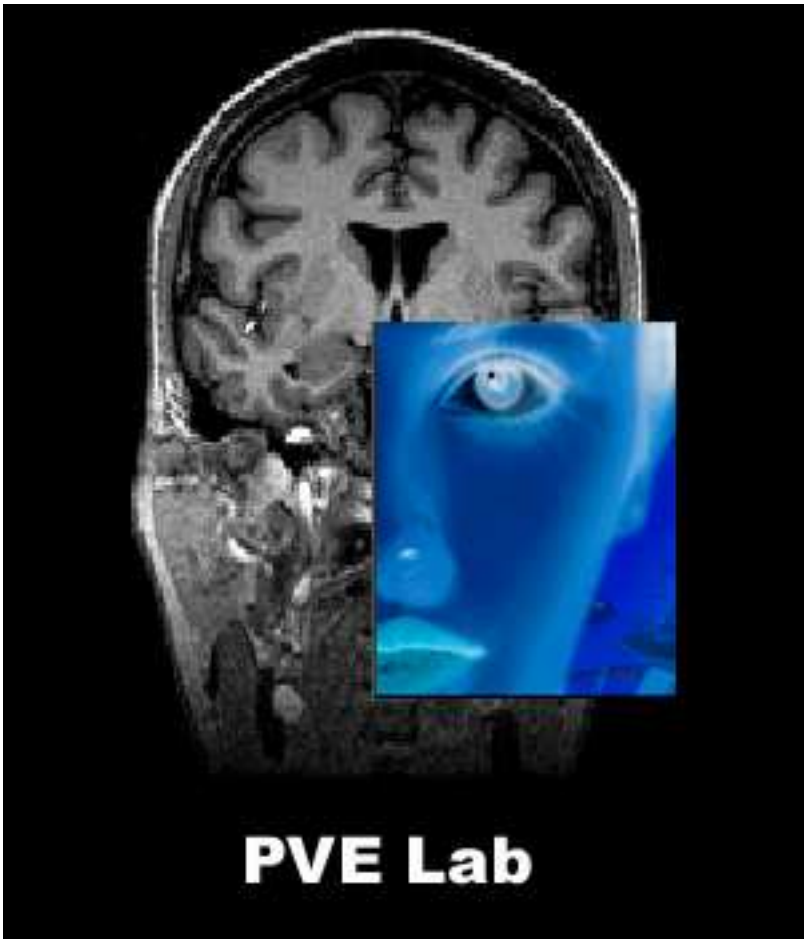
PVE Lab

1 The pipeline program

In the following a pipeline program is introduced. The pipeline program offers an easy way to set-up a pipeline containing one or more steps in a user-defined order. A step in the pipeline is refereed as a *task* and within each task there exist none or more programs each realising the given task. Such a program for a task is refereed as a *method* and could be any type of program: Matlab function, executable, Fortran and many more.

The pipeline program is based on a documented data structure where information of the individual pipeline is stored. The data structure can be seen as a read-/writeable database refereed as the project for the actual pipeline, section 4.4. For each project a unique working directory is automatically created, where the project and files generated through a pipeline are stored. In this way files within different projects are not mixed up, and setup-information can always be retrieved/repeated reading a project file.

Once a pipeline is setup an user-friendly graphical interface (GUI) gives a good overview and is easy-to-use special, designed for none-technical users, section 2. The GUI design of the pipeline program is homogeneous, in that way the GUI is very alike even for different types pipelines.
A pipeline can be executed without any user interfering (automatic) given that each method in the tasks are automatic.

To set-up a pipeline in the pipeline program (with GUI) only two steps must be done:

1) Make a setup file which consist of a few predefined fields describing used tasks, methods and their execution order in the pipeline. An easy-to-go guideline making a setup file see section 3.3, and an example, section 4.5.1.
2) Make a wrapper for each method to interface the pipeline program. Please read section 3 to understand the philosophy of the pipeline program. An easy-to-go example writing a function wrapper, see section 4.5.2.
   (advanced) Make a wrapper for a configurator, if users should have the possibility to setup a method within a pipeline. Please read section 3, An easy-to-go example writing a configurator wrapper, see section 4.5.3.

In the following all examples used referring to the PVELab pipeline, which is a part of the PVEOut project within the 5th European frame work. The EU project collects different methods in a pipeline (PVELab) to make a correction for partial volume error (PVE) of low resolution positron emission tomography (PET) brain scans using high resolution magnetic resonance (MR) scans. For more details go to homepage: http://pveout.area.na.cnr.it/.

1.1 System and Data Requirements

The pipeline program is running on workstations with Matlab version 6.5 or newer. The pipeline program itself take advance of the Matlab feature to be platform independent. It has been tested on Linux, Unix and Windows platforms. Please note that not all methods in the PVELab is platform independent.
Note regarding platform independence:

- Projects are not platform independent, because of difference in the path definition!
- Platform independence is only preserved if methods in a pipeline are!

The image data format within a pipeline is individual for the pipelines, but using the standard 'fileLoad_wrapper' the Analyze format (both 'ieee-le' and 'ieee-be') and DICOM is supported. The function can easily be extended.

2 The graphical user-interface

In this section the graphical user-interface of the pipeline program will be explained and demonstrated on the PVELab pipeline. Please remark that the user-interface is exactly the same for other pipelines except for names of tasks and methods.

2.1 Description of the user-interface

In Figure 1 is shown the PVELab, where input files are selected as showed in the title bar and the first task is done. The show button has just been pressed.

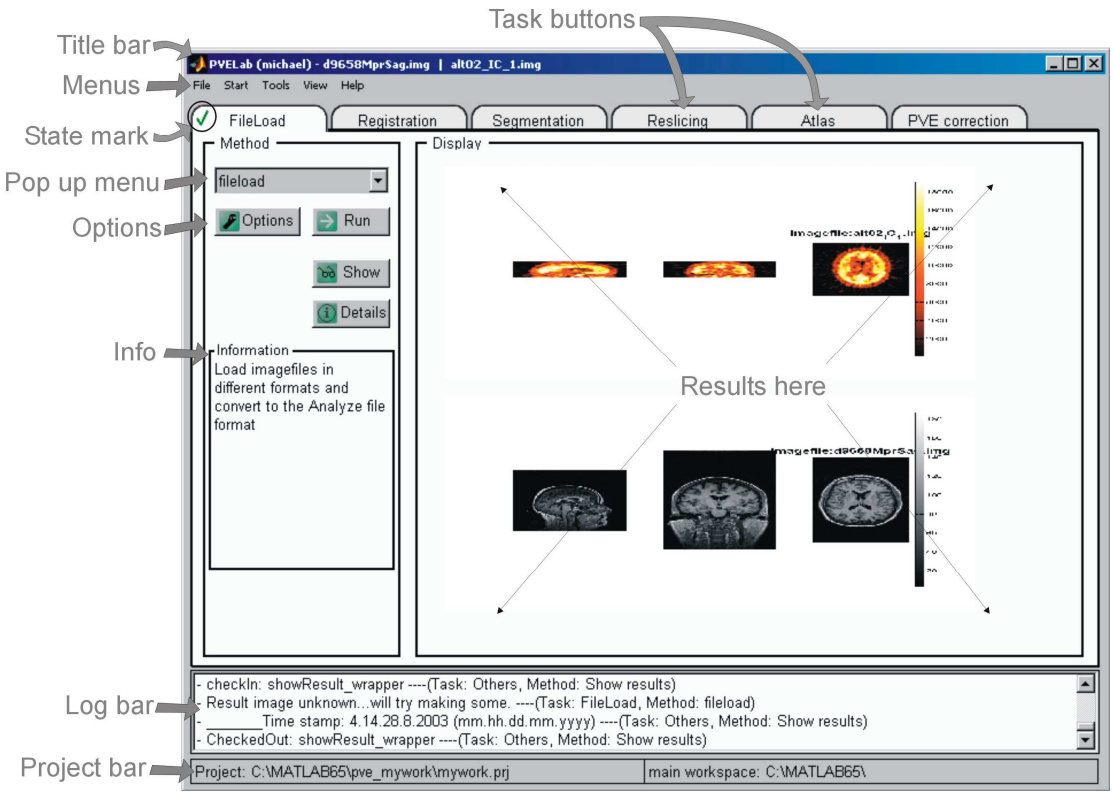

Figure 1

- **Title bar**: Name of the loaded pipeline is shown (PVELab), then username logged (username). When loaded (as in Figure 1) names of input brain images are also shown in the title bar.
- **Task buttons**: Name and order of tasks in the pipeline. A task is selected by pressing the actual task button. The selected task is highlighted (as FileLoad in Figure 1). Tasks in the PVELab are: Fileload, Registration, Segmentation, PVE correction, Atlas and Reslicing.
- **State mark** show the status of the individual task:

  → Task is active and running. No other tasks can run while a task is active.

  ✓ Task is finished with success. Next task can be selected and executed.

Task has detected an error. An error message is given. Project is cleaned-up by erasing generated files within the task and afterward ready to make a re-run. The task will indicate an error was detected until success.

- **Display**: Here progressing data can be presented to the user, but only if the given method support this feature. As shown, results from a task saved as a bitmap or jpeg, can be presented when the Show button is pressed.
- **Log bar**: All information logged in the pipeline are presented in the Log bar. The information is also saved in a log file with same name as the project files but with extension (*.log).
- **Project bar:** Show selected the main workspace, where newer project are stored in subdirectories, and name of active project in the pipeline if one is loaded or if a pipeline is started.

'Method' here all choices of a selected method include configuration within a task is given:

- **Pop up menu**: Select one of available methods within the selected task. When the pipeline program is started a default method is given. (project.pipeline.defaultPipeline)
- **Run button**: Executes only selected method within given task, but only if required tasks before in the pipeline are done. This button is always enable.
- **Show button:** When a task is done a snapshot of the output images (if they are available) or bitmap(s) given in the project can be shown in the 'Display'. This button is enabled when the task is done.
  (project.taskDone{TaskIndex,MethodIndex}.show)
- **Details button:** When a task is done used settings, selected method and generated files for the task can be listed in a separate window pressing by pressing the 'Details' button. This button is enable when task is done.
- **Information:** This field present a short information to the user about the selected method.
- **Options button:** Each method has the possibility to be configured by the user. If this possibility exist for a method the 'Option' button is enable. In Figure 2 is the Reslice configurator shown.

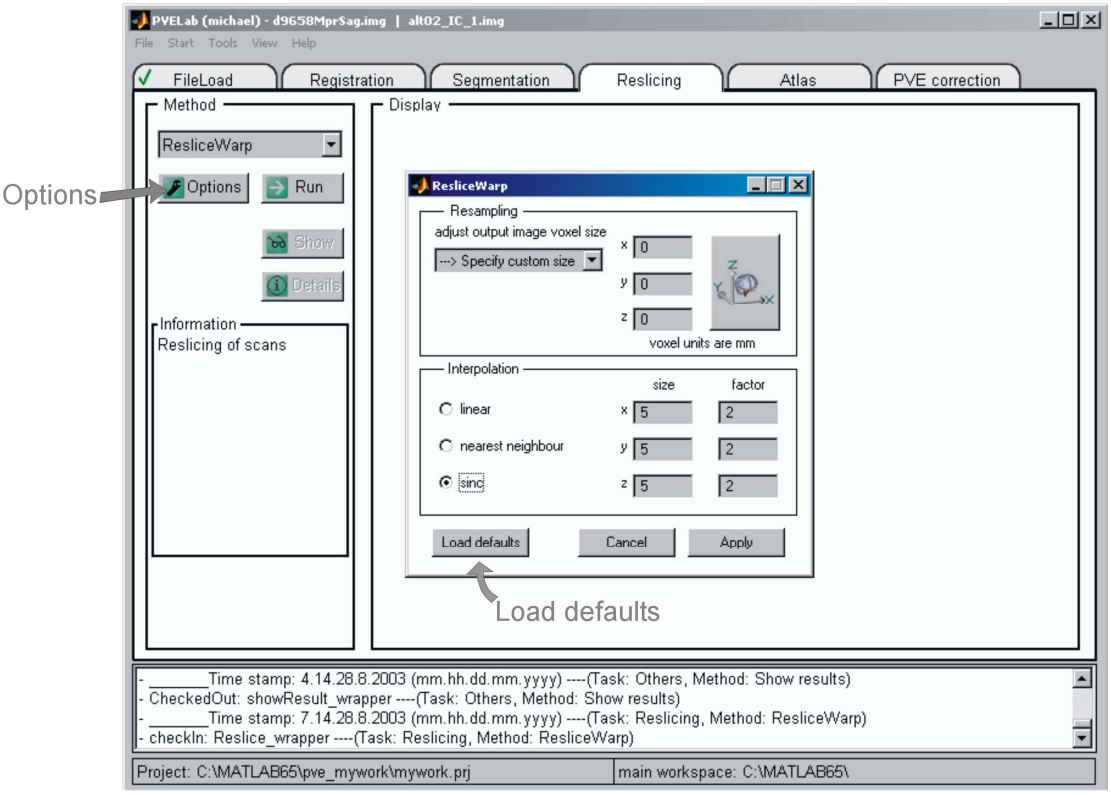

Figure 2. Option for the method ResliceWarp in the Reslicing task.

2.2 Menus

Below is given a description of the available menus in the PVELab pipeline. Menus not general/belong to the user-interface of the pipeline program is marked with (*).
For programmers making wrappers and updates to the pipeline program there will be referred to which field(s) in the project structure a given menu is using.

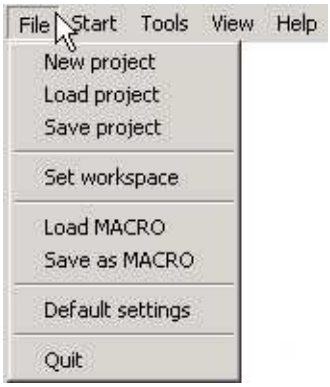

Figure 3 File menu in the PVELab

File menu handle setup of project files in a pipeline and where to save projects.

- **New project**: Close down the existing project, and reloads the pipeline program with default settings given in the setup file. The project is ready to use.

- **Load project**: Close down the existing project, and let the user choose a saved project (*.prj) from disc. The pipeline program is reset and initiated with setting given in the loaded project. The project is ready to use.
- **Save project**: Save/update existing project to disc. Note that the project is saved/updated automatic each time a task is finished.
- **Set workspace**: Change Directory to a main workspace where new project is saved in subdirectories. The main workspace can only be set once after starting the pipeline program or when choosing menu 'New project'. By default the main workspace is the directory from where the pipeline program is started. Note that the user must have write permission in the workspace.
- **Load MACRO**: Let the user selects a macro file (*.mac). Then close down the existing project and resets the pipeline program with the settings given in the macro. A MACRO (*.mac) contain user defined settings for both a given pipeline and configurations for methods in the pipeline.
- **Save MACRO**: Let the user selects a macro file (*.mac). All settings for the existing pipeline is saved into the macro file. Settings are both how the existing pipeline is setup, and how methods are configured. If default setting is chosen in existing project, then default settings is saved into the macro file.
- **Default settings**: Load default chosen methods for the each tasks in pipeline. Note that existing project is NOT reset and the configuration of the methods in the pipeline is unchanged. Default settings is found in the project (project.pipeline.defaultPipeline) and is found in the setup file string the pipeline program.
- **Quit**: Close down existing project, clean up and close GUI.

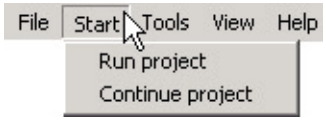

Figure 4, Start menu in the PVELab

Start menu handle execution of loaded pipeline with given settings.

- **Run project**: Sequential execution from first (left) to the last task (right). If one or more tasks already are done, they are all cleared and files are cleaned-up before the sequential execution. Note once started the sequential execution can not be stopped and the settings not be changed.
- **Continue project**: Continue sequential execution of a not finished project. Note once started the sequential execution can not be stopped and the settings not be changed.

Figure 5, Tools menu in the PVELab

Tools menu contain special tools and possibilities which is not a part of the pipeline. New tools can easily be added.

- New toll: A dummy menu. Is an example on how to add a new tool.

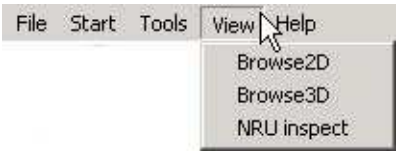

Figure 6, View menu in the PVELab

View menu contain different browsers to show different type of files. Other viewers can easily be added. Please note that the browsers expects the Analyze image data format.

- **Browse2D(*)**: Browse brain images in 2D given in the Analyze image format. In By default the Structural image of the firs task is automatic loaded in PVELab.
- **Browse3D(*)**: (More advanced than Browse2D) Browse brain images in 3D given in the Analyze image format. In By default the Structural image of the firs task is automatic loaded in PVELab.
- **NRU inspect(*)**: Advanced browsing of two brain images in view either as image overlay and as two images beside each other. The viewer has an extra feature to show a set of co-registrated images before reslicing if the co-registration matrix is found in the project.('project.taskDone{TaskIndex}.userdata.AIRfile')

Figure 7, Help menu in the PVELab

Help menu show documentation, references for available methods in the pipeline.

- **Documentation**: (NOT DONE): Here links to documentation of the available methods will be shown
- **About**: Show name and how did the Pipeline Program
- **Version**: (NOT DONE) Show version number of the pipeline program

2.3 Starting the PVELab in the pipeline program

Before starting the pipeline program:

- Add to Matlab path where system files of the pipeline program are placed. This should be done once.
- Change directory to where you like your a starting point.
- Starting the pipeline program given the setupfile *'setupPVELab.ini'* , then do:
  >setupPipeline(setupfilename).

The pipeline program starts by testing the setup file for tasks, order of the pipeline and that available methods exist. All this information are displayed into the Log window, Figure 1.
Example of a started pipeline, PVELab, is shown in Figure 8.
Follow the message boxes to select, setup and run a method within a selected task.

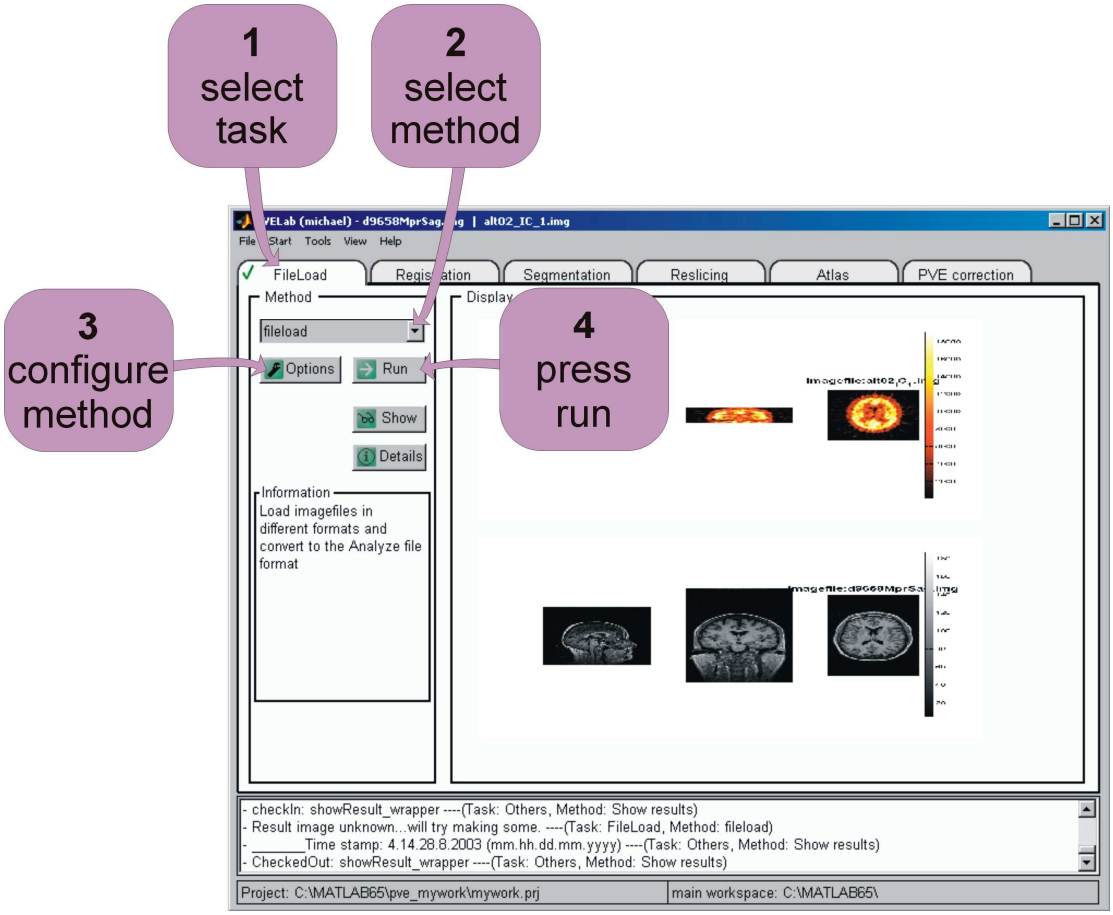

Figure 8. Example of starting a task….

3 Making a pipeline

As introduced the pipeline program is a flexible collection of different methods in a pipeline and offers a general and easy-to graphical user interface. Before this is a reality a pipeline must be setup and the implemented methods/programs must interface the pipeline program. A *function wrapper* is used to interface between the pipeline program and individual methods. Methods giving a user the possibility to configure given method, should also use a *configuration wrapper* (this wrapper is optional). When all wrappers are done and ready, a setup file configuring the pipeline in the pipeline program can be made.

Please note that both making wrappers and setup file is not a complicated task: Just use available guidelines and examples, but also read this section to get a feeling of how wrappers and setup files are made and work!

3.1 Function wrapper

The function wrapper is responsible to make the pipeline program general in a way so that different methods can be implemented without conflicting. It is typically called by the main_wrapper (section 4.1.2), but can also be called by another function wrappers. Some methods are very alike in the way the are called so the same function wrapper can be used. It is also possible to implement a method directly in the function wrapper. In that case the method do interface the pipeline program.

Restrictions making a function wrapper are those for reading/writing the project (see section 4.4.1), and that some input and output arguments are fixed:

Syntax interfacing a function wrapper to the pipeline program:

[project,varargout] =MethodName_Wrapper(project,TaskIndex,MethodIndex,varargin)

Where *project* is the data structure and the indices are refereed to the field of the project with the set-up for the actual method given in *project.pipeline.taskSetup{TaskIndex,MethodIndex}* (section 4.4.4). *Varargin* and *varargout* holds an arbitrary number of user input, output arguments. Please note that these parameters are only understood by the given function and is not a part of the pipeline program.

In section 4.5.2, Figure 17 and Figure 18 is shown an example on how a function wrapper is made. The example can be used as a starting point making a new function wrapper. Just rename function name and filename (must be the same). For clearness please keep the suffix '_wrapper'.

3.2 Configurator wrapper

A configurator wrapper is responsible for setting up a user-interface so users can select and/or change settings of a given method, and update the new user defined settings in the project. The configurator wrapper is activated when the user press the 'options' button for a given method (Figure 2). New settings will be used next time the method is executed.
Note that

- The configurator is optional.
- It is good practise that a user always can select default settings.

Syntax interfacing a configurator wrapper to the pipeline program:

[project,varargout] =NameConfig_Wrapper(project,TaskIndex,MethodIndex,varargin)

Where *project* is the data structure and the indices are refereed to the field of the project with the set-up for the actual method given in *project.pipeline.taskSetup{TaskIndex,MethodIndex}* (section 4.4.4). *Varargin* and *varargout* holds an arbitrary number of user input, output arguments. Please note that these parameters are only understood by the given function and is not a part of the pipeline program.
In section 4.5.3 and in Figure 19 is shown an example on how a configurator wrapper is made. The example can be used as a starting point making a new configurator wrapper.

Just rename the function name and filename (must be the same). For clearness please keep the suffix 'Config_wrapper'.

3.3 The setup file (.ini)*

The setup file is a text file with the extension (*.ini). It is responsible to setup tasks, their order in the pipeline and available methods within each task.
Making a setup file is actually just filling in predefined fields in a data structure for each method. The Pipeline Program loads the setup file into 'project.pipeline.taskSetup{TaskIndex,MethodIndex}', section 4.4.4. Please note that only predefined fields are accepted by The Pipeline Program.

Making a setup file for The Pipeline Program:
1) Make a sketch of tasks to be included.
2) For each method fill in a guideline for the setup file given in section 3.1.
3) Create a setup file using 'ExampleLab.ini' as template and information given from step 2.
4) Save the setup file with extension (*.ini).
5) Ready to test setup file as in section 2.3.

Example (*ExampleLab*) of making a setup file section 4.5.1 and Figure 16.

3.3.1 Guideline for the setup file

The below fields (shown in bold) should be fill in for each method as preparation making a setup file. An explanation for each field is given and examples used for the PVELab pipeline are found in section 4.2.

- **Method:** Tag name of method name to appear in the pipeline program.

- **Method_name**: Full name of method.

- **Task:** Name of the task the given method belongs to/ is realising.

- **Version label**: A text string to identify the version number of the method in a Matlab file.

- **Description**: Information given via the information window to the user about selected method and its special features.

- **Documentation**: Article or where (e.g. www) the method is published.

- **Who did**: Who did the method (group), city and year.

- **Require task index**: Tasks to be done before method may run.

- **Function name:** Name of the function that is realising the method. The function is executed within the Matlab environment. The same name for the function and method can be used.

- **Function syntax**: Syntax for the function.

- **Input arguments**: Description of the input parameters.
- **Output arguments**: Description of the output parameters.
- **Code type**: Matlab, executable or like to be executed from Matlab.
- **Configuration default**: Default settings for method. Is not changed.
- **Configuration name**: If a function is used same principal as function name

3.4 A project (.prj)*

As introduced each project contain a data structure where all information for a pipeline are stored. Information regards settings for the tasks and available methods in the pipeline, results, log information and status of the pipeline process. There exist a project for each started pipeline which can be loaded and saved when needed.
A description of the fields available in the project structure is given in section 4.4.

4 Appendix

4.1 Connection diagram of The Pipeline Program

In the following the connectivity between user-interface, wrappers and files in the pipeline program are explained and referenced to Figure 9.

Robustness, transparency and generality are key words of the pipeline program.

- Robustness: The pipeline program is not affected by a crashing method.
- Transparency: All necessary information regards set-up, used settings and log information are automatic collected and are fully available within the pipeline program.
- Generality: The pipeline program is easy to use, independent of operation systems, and different type of executable realising a method can be used.

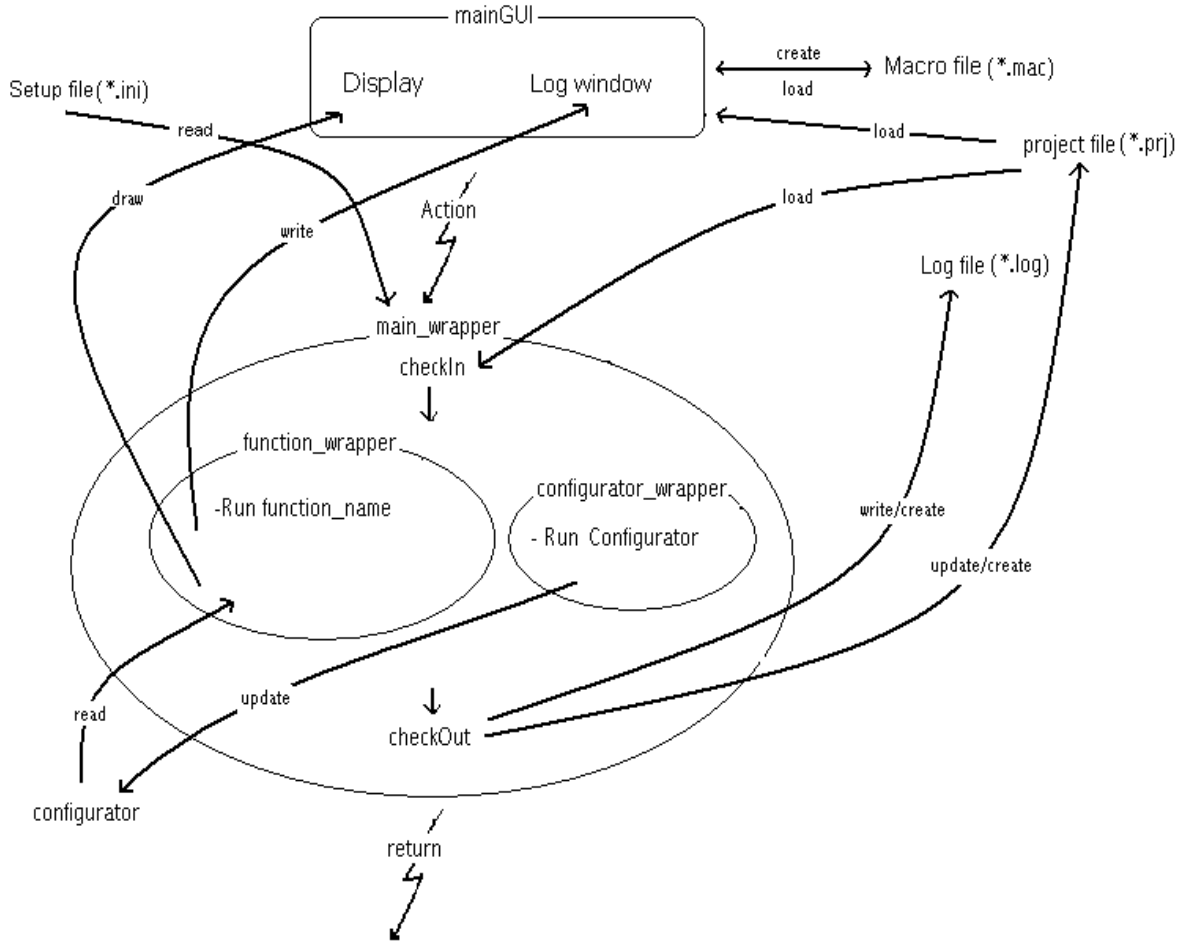

Figure 9 A connection diagram of the pipeline program showing the connection and flow between the different part of the pipeline program and used files.

4.1.1 Function and configurator wrapper

A wrapper is a function, which interface a method to/from the pipeline program. Basically two type of wrappers exist within the pipeline program: the function_wrapper (section 13.1) and the configurator_wrapper (section 3.2).
The basic idea of the wrappers is to make the pipeline program generel so a vary of different methods can be interfaced and act as a unit.

4.1.2 The main_wrapper

The robustness of the pipeline program is insured by the main_wrapper, it control the flow in a pipeline. Further it handles the execution of a method and error handling if a method crashes. It is a generel function with information of the different programs only given by the setupfile placed in project.pipeline.taskSetup{TaskIndex,MethodIndex}, section 4.4.4.

4.1.3 checkIn and checkOut

In a pipeline it is important to control process so e.x. that two methods can not be executed at the time and which methods is done. Here to functions are used.

'checkIn' checks in a method into a restricted area. When checking in, the status of the pipeline is controlled, and if the selected method is allowed to be started. Further is checked if the program of the method exist as given in the setup file. If OK the method is checked in. The project.taskDone{TaskIndex} (section4.4.5) is created and the method is ready for execution. If an error is detected a message is given and the method is not started.

'checkOut' checks out a method from the restricted area and free it for the next task. It checks if an error has occurred. If it is the case all files generated within the actual task is deleted and the project is cleaned up and project.taskDone{TaskIndex} is removed from the project.

4.1.4 The mainGUI

The user-interface is handled by a function called *mainGUI* shown in section 2. Here handles to the Display and the Log window are given, in this way progressing data, results and log information can be presented for the user. The pipeline program could in principal be executed without the mainGUI.

4.2 Guideline: Prepare setup file for PVELab

4.2.1 Co-registration: IIO

Method: IIO
Method name: Interactive Image Overlay
Task: Co-registration
Version label: version
Description: Interactive Image Overlay (IIO) a manual method for co-registration (6-DOF) high resolutions MR scan and a low resolution PET/SPECT brain scan.
Documentation:
Who did: NRU, Copenhagen, 2000-2003
Require task index: 1 (Files load)
Function name: coreg
Function syntax: coreg('*batch*',[],*Filename*,*ParentHandle*,*ReturnFucn*)
Input parameters:

Filename: structure containing a high resolution MR scan and one or more low resolutions PET/SPECT given in a cell array. All in the Analyze image format.

Example of *Filename* structure:
Filename.RES: Name and path for high resolution MR scan
Filename.STD{}: Name and path of one or more low resolution PET/SEPCT scans

Filename.A: If exist, a given co-registration matrix is used as initiated co-registration
Filename.AIR: If filename exist, co-registration matrix is saved in air-format.

ParentHandle: Figure handle to parent/main figure so it can be found when returning fra Coreg.
Example:
ParentHandle=h_mainfig

ReturnFucn: Function to call exiting the Coreg program.
Example when exiting from Coreg in Matlab:
feval(ReturnFucn,'ReturningData',ParentHandle,Aall);
where
- *ReturnFucn='checkOut'*
- '*ReturningData'*: Parameter telling that co-registration matrix is returned by Coreg.
- *ParentHandle=h_mainfig.*
- *Aall* cell array contain co-registration matrix.

Output parameters: None, is using the '*ReturnFucn*'
Code type: Matlab code, (figure object)
Configuration: none

4.2.2 Co-registration: IPS

Method: Interactive Point Selection
Method short-term: IPS
Task: Co-registration
Version label: SWlable
Description: Interactive Point Selection (IPS) a manual method for co-registration (6-DOF) a high resolution MR scan and a low resolution PET/SPECT brain scan. User mark a least 3 different sets of corresponding points in the MR and the PET/SPECT scan. A co-registration matrix is found based on the point set. Extra features: Save/load point sets and images. Reslicing and visualisation of the co-registration.
Documentation:
Who did: NRU, Copenhagen, 2000-2003
Require: Files load
Function: registrate
Function syntax: *registrate*('*batch*',[],*Filename*,*ParentHandle*,*ReturnFucn, Visualizer*)
Input parameters and syntax:
Filename: structure containing a high resolution MR scan and one or more low resolutions PET/SPECT scans given in a cell array. Both images saved in the Analyze format.
Example of *Filename* structure:
Filename.RES: Name and path for high resolution MR scan
Filename.STD{}: Name and path of one or more low resolution PET/SEPCT scans
Filename.A: If exist, a given co-registration matrix is used as initiated co-registration.
Filename.AIR: If filename exist, co-registration matrix is saved in air-format.

ParentHandle: Figure handle to parent/main figure so it can be found when returning from Registrate.
Example :
ParentHandle=h_mainfig

ReturnFucn: Name of function where to *checkout* before exiting the Registrate program.
Example when exiting from Registrate in Matlab:
feval(ReturnFucn,'ReturningData',ParentHandle,Aall);
where
ReturnFucn='checkOut'
'*ReturningData'*: Parameter telling that co-registration matrix is returned by Registrate
ParentHandle=h_mainfig
Aall cell array contain co-registration matrix
Visualizer: Method to inspect the co-registration. Default: 'inspect'.

Output parameters:
Code type: Matlab code, (figure object)

4.2.3 Viewer: Browse2D

Method: Browse2d
Method short-term: Browse2D
Task: Viewers
Version label: version
Description: A 2D viewer where one or more images simultaneous can be inspected and compared.
Documentation:
Who did: NRU, Copenhagen, 2000-2003
Require: Fileload
Function: browse2d
Function syntax: browse2d('LoadData', [], Filename)
Input parameters:
'LoadData': Command text string (default)
Filename: Text string containing path and name of an image to be loaded in the Analyze format.
Output parameters:
Code type: Matlab code, (figure object)
Configuration default: 'Batch'.

4.2.4 Viewer: Browse3D

Method: Browse3d
Method short-term: Browse3D
Task: Viewers
Version label: version
Description: A 3D viewer where one or more images simultaneous can be inspected and compared. Extra features: Header information upon the loaded brain. Possible to do a coarse alignment of current image (have no meaning in this situations). Possible to show contour with a user selected threshold.
Documentation:
Who did: NRU, Copenhagen, 2000-2003
Require: File load
Function: browse3d
Function syntax: browse3d=*('LoadData'*, [], *Filename)*
Input parameters:

'LoadData': Command text string
Filename: Text string containing path and name of image to be loaded in the Analyze format.
Output parameters:
Code type: Matlab code (figure object)
Configuration default: 'Batch'

4.2.5 Viewer: NRU inspect

Method: Inspect
Method short-term: Inspect
Task: Viewers
Version label: version
Description: A 2D viewer where two images (typ.: MR and PET/SPECT) can be inspected and compared either by transparency overlay or listed beside each other. Extra features: Run and save a movie frame of cuts through all or parts of the brain in either x-, y- or the z-direction.
Documentation:
Who did: NRU, Copenhagen, 2000-2002
Require: Load files
Function: inspect
Function syntax: inspect=*('LoadData', Filename)*
Input parameters and syntax:
'LoadData': Command text string
Filename: Structure of text strings containing path and name of image to be loaded in the Analyze format. OR a sturcture of loadede imagedata and header information.
example of a Filename structure
Filename.imgPET: Imagedata or name and path for PET/SPECT scan
Filename.hdrPET: Headerdata or name and path for PET/SPECT scan
Filename.imgMR: Imagedata or name and path for MR scan
Filename.hdrMR: Headerdata or name and path for MR scan
Output parameters:
Code type: Matlab code, (figure object)
Configuration: 'Batch'

4.3 File formats and directories

A short description on files and directories used by the Pipeline Program.

- **The pipeline Program**: Matlab search path is set-up with sub directories to the system file '*setupPipeline.m*'. Added directories are removed on exit of the Pipeline Program.
- **Workspace**: unique project directory
  - Path: sub directory to *project.sysinfo.mainworkspace* with prefix given in the setup file. If directory exist an index is added to the prefix.
- **Log file**: Text file with log information of a project
  - Filename: same as project with extension '*.log' *project.sysinfo.logfile.filename.*
  - Path: *project.sysinfo.workspace*
- **Project file**: Matlab data file.
  - Filename: *project.sysinfo.prj_filename* with extension '*.prj'.
  - Path: *project.sysinfo.workspace*
- **Setup file**: A textfile setting up the tasks and methods in a pipeline
  - Filename: *project.pipelinesetupTask{end,1}.configurator. configurator_name* with extension '*.ini'.
  - Path: Within the Matlab search path
- **Macro file**: Matlab data file, with user defined setup of a pipeline
  - Filename: '*.mac'.
  - Path: user defined
- **Output files:** Generated image filenames of image data.
  - Filename: *project.pipeline.inputfile.filename{1,1}* add prefix given in the set-up file.
  - Path: *workdir*
- **Function_wrappers, configurator_wrapper:** Interfacing the pipeline program.
  - Filename: Given in the set-up file.
  - Path: Within the Matlab search path
- **Method files:** Realising a task
  - Filename: Given in the set-up file.
  - Path: Within the Matlab search path

4.4 Description of the Project structure

In the following sections a short description is given of the data structure of a project.

4.4.1 Restrictions using project

The data structure of a project is in general a READ-ONLY structure, unless else is given. A pipeline in the pipeline program is a collection of different tasks and methods implemented by different programmers. It is therefore obviously, not respecting given restrictions may cause to instability, not generality and misunderstanding using the pipeline program.

The data structure is therefore transferred between functions, each making a copy of the project. This can make the pipeline program slow and using unnecessary memory if lot of data is stored in the project. There fore do not store large data amounts in the project such as image data.

4.4.2 Syntax and indices

To navigate in a project one must understand the basic syntax of a data structure and used indices.

A data structure is build up of blocks containing one or more fields of those illustrated in Figure 10.

Type of fields:
- field0: Contain one or more given type of fields: field0 to field4.
- field1: One variable of type string, number or a field.
- field2(index): Vector a specific type of variables, such as numbers or strings with the same length.
- field3{index}:Cell array of none-specific type of variables or fields.
- field{index1,index2}: As field3, but with an extra dimension.

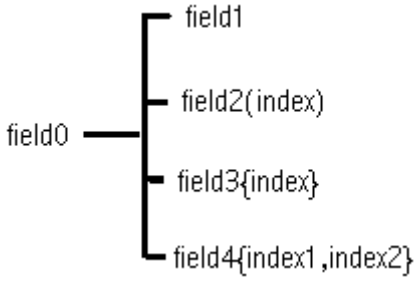

Figure 10 Example of fields used in the pipeline program.

To access field1 to field4 in structure field0 do:
Reading a field into variable A:

A=field0.field1;
A=field0.field2(index);

Writing variable to a field:

field0.field1=A;
field0.field2(index)=A;

Indices used for navigating a project in the pipeline program:

- **Index**: not a specific index.
- **TaskIndex**: Selected task of a defined order in the pipeline. First task is 1 and is placed left in the user-interface.
- **MethodIndex**: Selected method of available methods in a task: First method is 1, and placed highest in the pop-up menu in the user-interface .
- **ModalityIndex**: Selected modality of available modalities: First modality is 1
- **ImageIndex**: Selected image of available images: First image is 1 (default).

4.4.3 The project field

The project has four fields each containing a structure and is explained in the following sections.

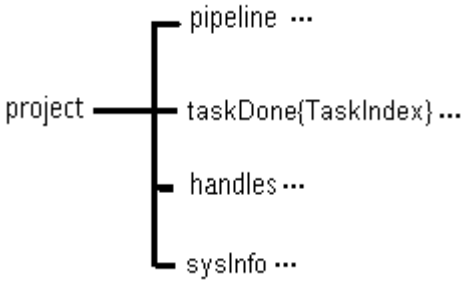

Figure 11 Basic fields in the project structure

Fields of the project:

- **pipeline**: Contain both settings for the pipeline and status information of the pipeline process.
- **taskDone{TaskIndex}**: Information of a finished task in the pipeline process.
- **handles**: Handles to available figures in the pipeline program.
- **sysInfo**: General system information such as directories and files.

4.4.4 The pipeline field and sub-fields

The pipeline field contain all information regarding setup, control and status of the pipeline. The field *taskSetup{TaskIndex,MethodIndex}* is a sub-field of settings for the given method of a task. Here is the information from the setup file placed when loading a new project or starting the pipeline project.

The pipeline field is shown in Figure 12 and a short description of each field is given in the following.

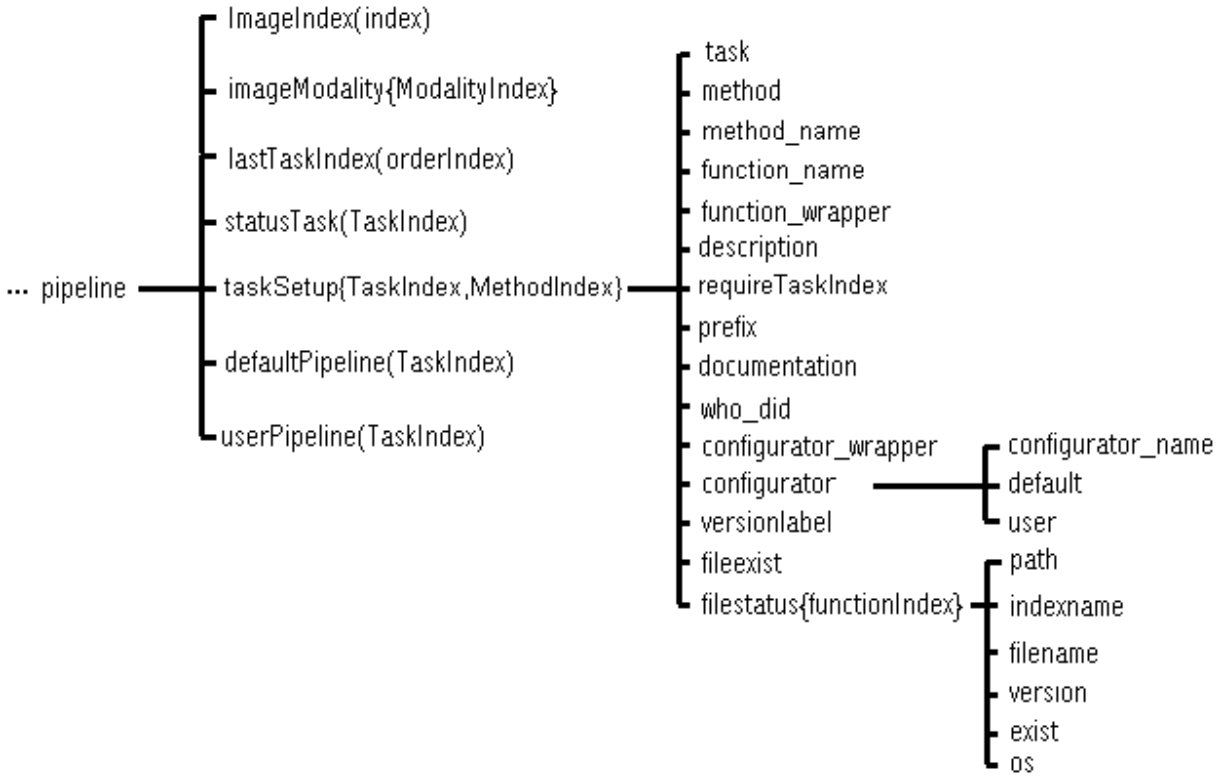

Figure 12 Fields of pipeline found in the project structure

Fields of pipeline:

- **LastTaskIndex(Index)**: Vector containing the index to the last succeeded task in an increasing order.
- **StatusTask(TaskIndex)**: Vector containing a status-flag of each task: 0= Ready, 1=Active/runing, 2=Done (if succeed) and 3=Error has been catched.
- **DefaultPipeline(TaskIndex)**:Vector containing index to the method which is by default chosen when the pipeline program is started. This is given in the setup file.
- **UserPipeline(TaskIndex):** Vector containing index to user selected methods in the pipeline.
- **taskSetup{TaskIndex,MethodIndex}**: Cell-array containing a sub-entry to a setup structure of each method in a task.
  - **task**: Task that method belongs to. The task name will appear in the pipeline program.
  - **method**: Name of the method (e.g short-term) to appear in the pipeline program.

- **method_name**: Full name of method.
- **function_name**: Program/function realising the method.
- **function_wrapper**: Wrapper that interface method and the pipeline program.
- **description**: Information of method, will appear in the pipeline program (information window).
- **requireTaskIndex**: Index of which tasks has to be finished before current task.
- **prefix**: Added to output files of actual task.
- **documentation**: Text or link how to get further documentation of given method. Will appear in 'about menu' of the pipeline program.
- **who_did**: Name of group and year. Will appear in 'about menu' of the pipeline program.
- **configurator:** (if not empty) text file with settings for actual method
    - **default**: Defult settings of a method.
    - **user**: User selected setting of a method.
    - **configurator_name**: Configurator function (*.m,*.exe or alike) called by the configurator_wrapper.
- **configurator_wrapper**: (if exist) Wrapper to interface a configurator and the pipeline program.
- **versionlabel**: Text label identifying version number of given method in a matlab file.
- **fileexist**: Ready flag if all needed functions in the task field are found by the pipieline program. 1=exist , 0=do not exist and method can not be executed.
- **filestatus{functionIndex}**: File status of functions for a given method in the current field of tasks{,}, which should be available to the pipeline program.
    - **path**: Where function is found
    - **indexname**: Field name
    - **filename**: Name of file for the actual function.
    - **version**: Found version label.
    - **exist**: If file exist = 1 else = 0.
    - **os**: Actual operation system

4.4.5 The taskDone field and its sub-fields

Information for each finished task in the pipeline is stored in this field. This regards from selected method, settings and program files. If an error is detected the field is cleaned up and all generated files, except input image files and program files are deleted.

In the field 'userdata' additional user information can be written either into a variable or a sub structure. All other fields are read-only.

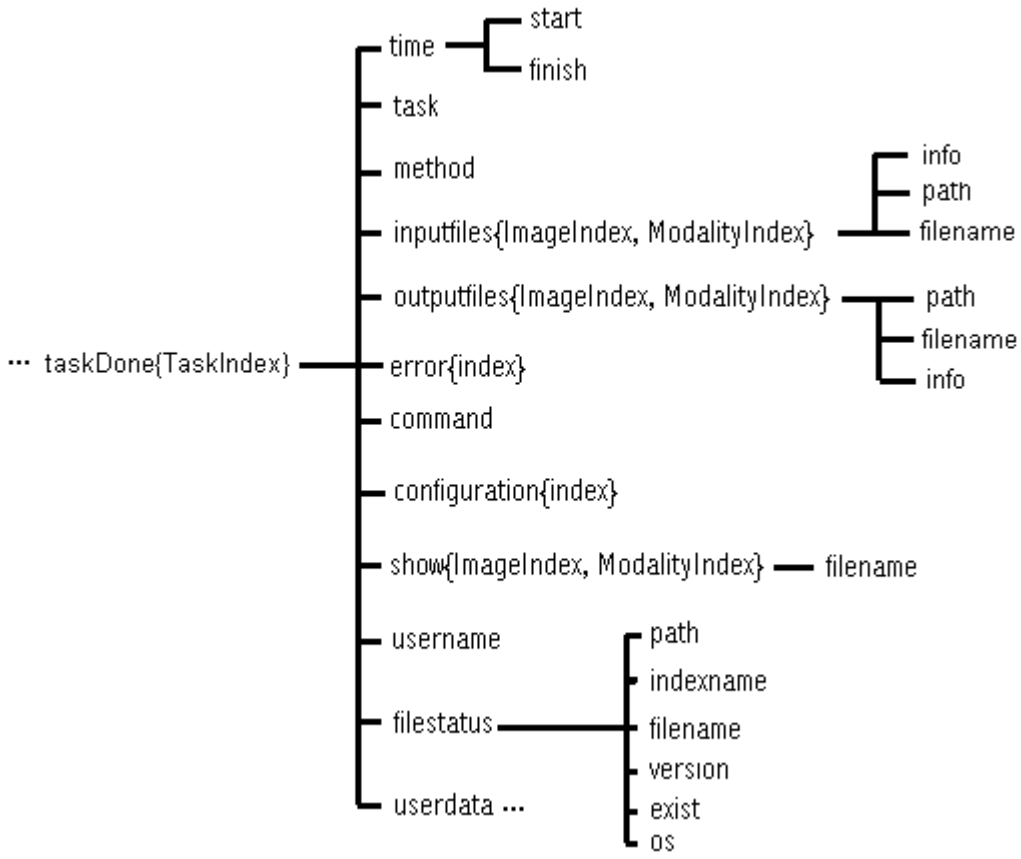

Figure 13 Fields of taskDone found in the project structure

Fields of taskDone{TaskIndex}:

- **Username**: user logged on the computer.
- **time**: Evaluation time for given method.
  - **start:** Start time(yy.dd.hh.mm.ss)
  - **finish:** End time (yy.dd.hh.mm.ss)
- **task**: Name of task given in: *project.pipeline.{,}.task*
- **method**: Tag of selected method given in: *project.pipeline.{,}.method*
- **inputfiles{ImageIndex,ModalityIndex}** (filenames are automatic made before the method is executed)
  - **filename**: header (*.hdr) or image (*.img) filename of input image, given a modality. Expected image format: Analyze.
  - **Path:** path of input image file of given modality.
  - **Info:** (optional user information) Additional text information of image.
- **outputfiles{ImageIndex,ModalityIndex}**:(file names are automatic made)

  - **filename filename**: header (*.hdr) or image (*.img) filename of output image, given a modality. Expected image format: Analyze
  - **path**: path of output image file of given modality.
  - **Info:** (optional user information) Additional text information of image.
- **error{index}:** cell-array of detected errors. Detected errors will clean up the field and delete generated files. For a finished task the error field is empty.
- **command**: A commands given to the method
- **configuration:** Used configuration parameters: NOT READY/IMPLEMENTED.
- **show{ImageIndex,ModalityIndex}**:(file names are automatic made) Show snapshot of the output data.
  - **filename**: Images (*.bmp,*jpg or like) that gives a snapshot of the output result from given method.
    Note: The images are shown in the Display of the pipeline program. These images are not made by the pipeline program, but by the programmer for the given method.
- **filestatus{functionIndex}**: File status of functions for a given method in the current field of tasks{,}, which should be available to the pipeline program.
  - **path**: Where function is found
  - **indexname**: Field name
  - **filename**: Name of file for the actual function.
  - **version**: Found version label.
  - **exist**: If file exist = 1 else = 0.
  - **os**: Actual operation system: Linux, Windows and ect.
- **userdata**: A user defined entry for the actual method. Here additional information can be stored as sub-entries the user needs or like to handle to the following tasks in the pipeline.

4.4.6 The handles field

Handles to axes that give the possibility for a method to present the user for progressing data or show images in the user interface of the pipeline program. If the pipeline program is executed without the user-interface the handles will not be available but empty.

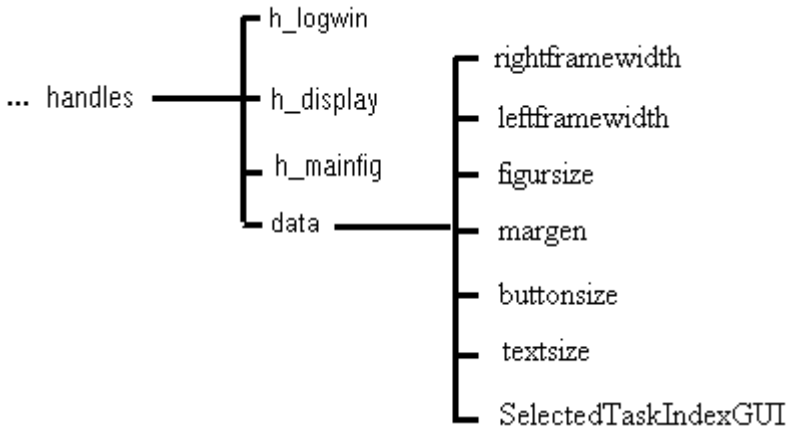

Figure 14 Fields of handles found in the project structure

Fields of handles:

- **h_logwin:** Handle to the log window where log information are presented to the user.
- **h_display**: Handle to the Display, where progress data and images of results can be presented to the user.
- **h_mainfig**: handle to the main figure of the user-interface, called mainGUI.
- **data:** Data used by the mainGUI, but is not saved into the project file.
  - **rightframewidth: (M. Twardak)**
  - **leftframewidth**: **(M. Twardak)**
  - **figursize**: **(M. Twardak)**
  - **buttonsize**: **(M. Twardak)**
  - **textsize**: **(M. Twardak)**
  - **SelectedTaskIndexGUI**: TaskIndex of selected task in the user-interface of the pipeline program.

4.4.7 The sysinfo field

In this field system information for the loaded pipeline and for the pipeline program are found.

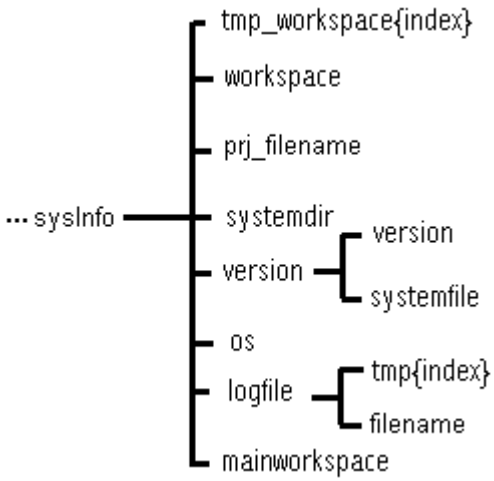

Figure 15 Fields of sysinfo found in the project structure

Fields of sysinfo:

- **mainworkspace**: Where new project are saved as subdirectories.
- **version:**
  - **systemfile:** filename used as system file default: '*setupPipelin.m*'
  - **versionlabel:** Version label of system file.
- **tmp_workspace:** Temporary directories added to Matlab search path, which are removed on exit.
- **os**: Operation System the project is created on. Possible states: WINDOWS, UNIX.
- **workspace**: Directory where all files generated in a pipeline process are saved.
  Note: A *workdir* is automatically created for each new project when the first task in the pipeline is finished.
- **systemdir**: Directory where system files for the pipeline program are placed.
- **Prj_filename**: filename of project with extension *'*.prj'*, saved in *workdir*.
- **version**: version number of the loaded pipeline set-up file.
- **logfile**: A log file exist for each project, and saved in the *workdi*r with extension *'*.log'*. Log information are transferred to the log bar via *project.handles.h_log* if the user-interface is used.
  - **tmp{index}:** Temporary cell-array of log data. Appended to the log filename when a method has finished.
  - **filename**: name of log file with extension *'*.log',* saved in *workdir*.

4.5 'ExampleLab' an example using The Pipeline Program

In the following section an example is given on making

- a setupfile called '*ExampleLab.ini*', section 4.5.1, which is setting up The Pipeline Program as shown in Figure 16.
- a function_wrapper called '*exampel_wrapper*' is a method that can be used in all type of tasks. It has three functions:
  1) Loading and show the start up bitmap (…/pili/mainGUI/bmp/wellcome.bmp) in the Display for 3 seconds as in Figure 17.
  2) Load configurator settings for actual method (number of progressing iterations) and show progressing data for the given number of iterations as in
  3) If a prefix for current method is given in the setup file, then load image files and save then with given prefix.
- a configurator wrapper called '*exampleConfig_wrapper*' is an example making a configurator with an user-interface and read/updating settings for given method.

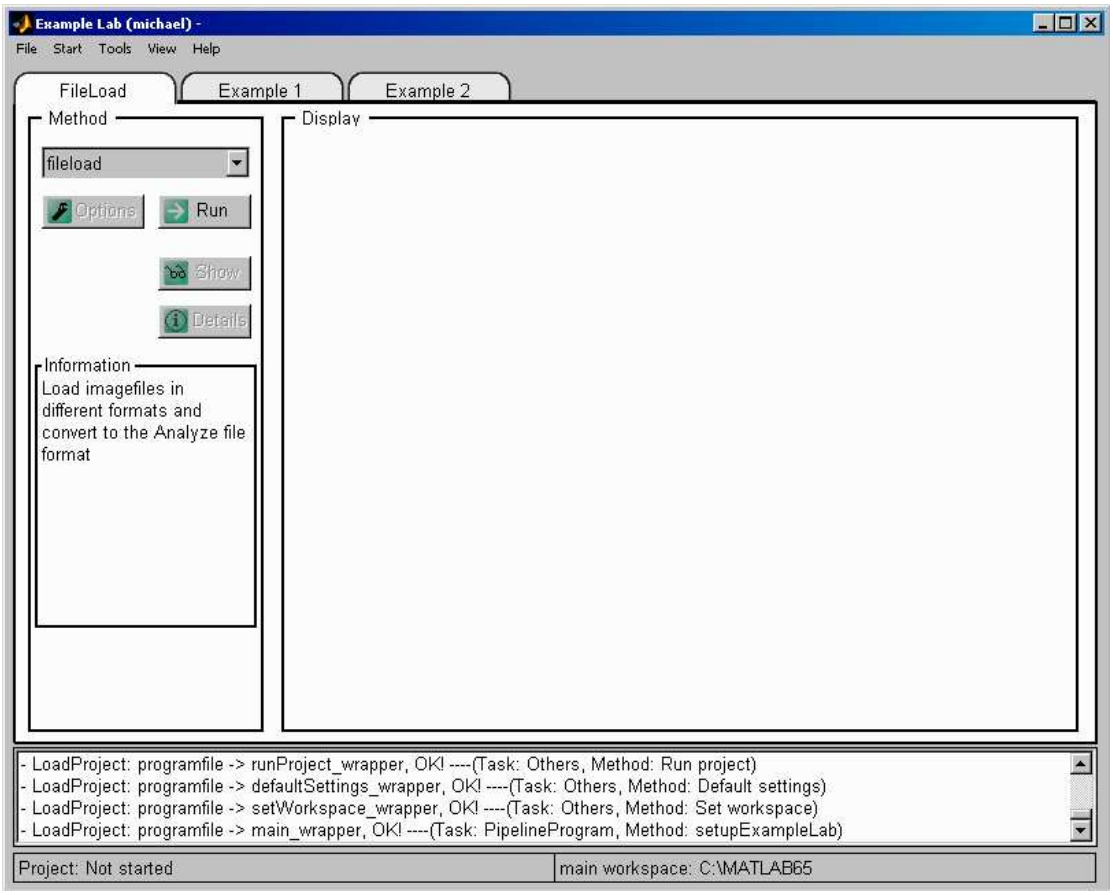

Figure 16 Upstart of ExampleLab in the Pipeline Program using the setupfile 'ExampleLab.ini'

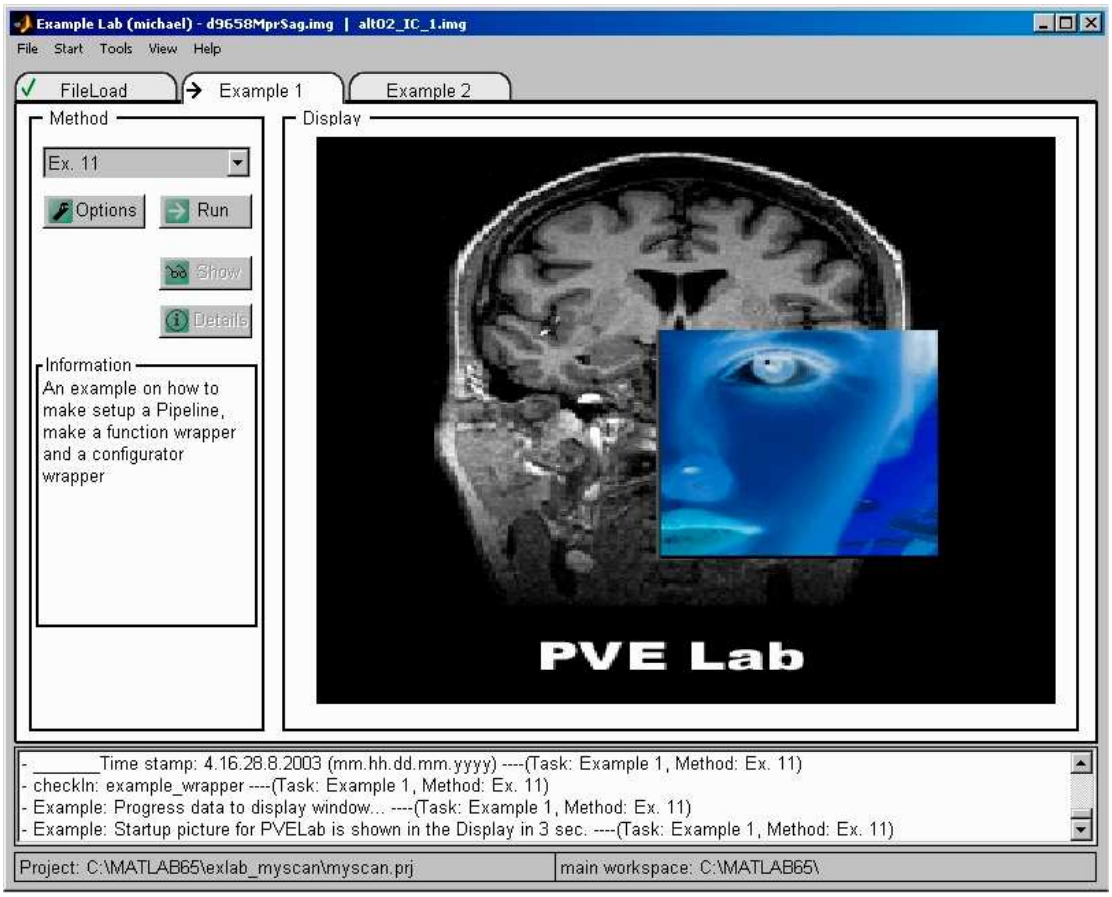

Figure 17 'RUN' button has been pressed in ExampleLab, the first the welcome picture is showed for 3 seconds.

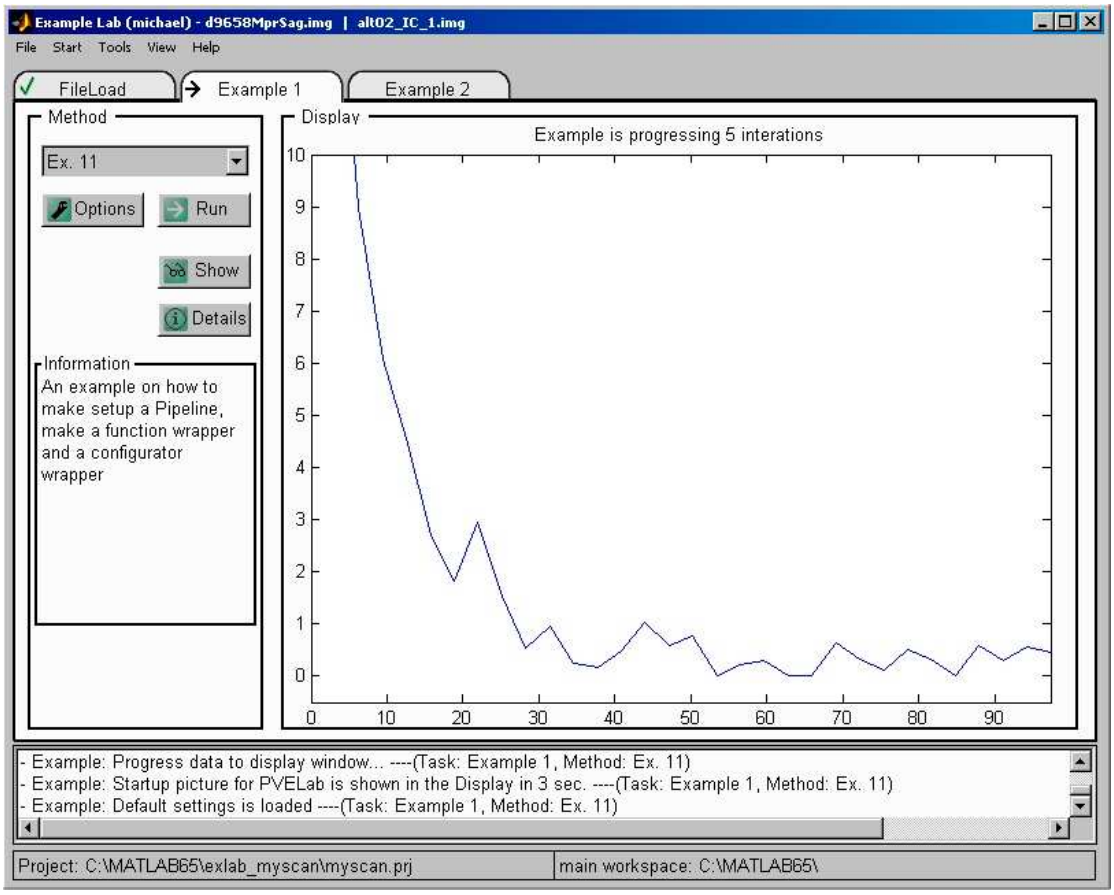

Figure 18 ….'RUN' button has been pressed….next progressing data is showed in the 'Display' for a number of iterations as given in settings. The user can change this pressing 'OPTIONS'.

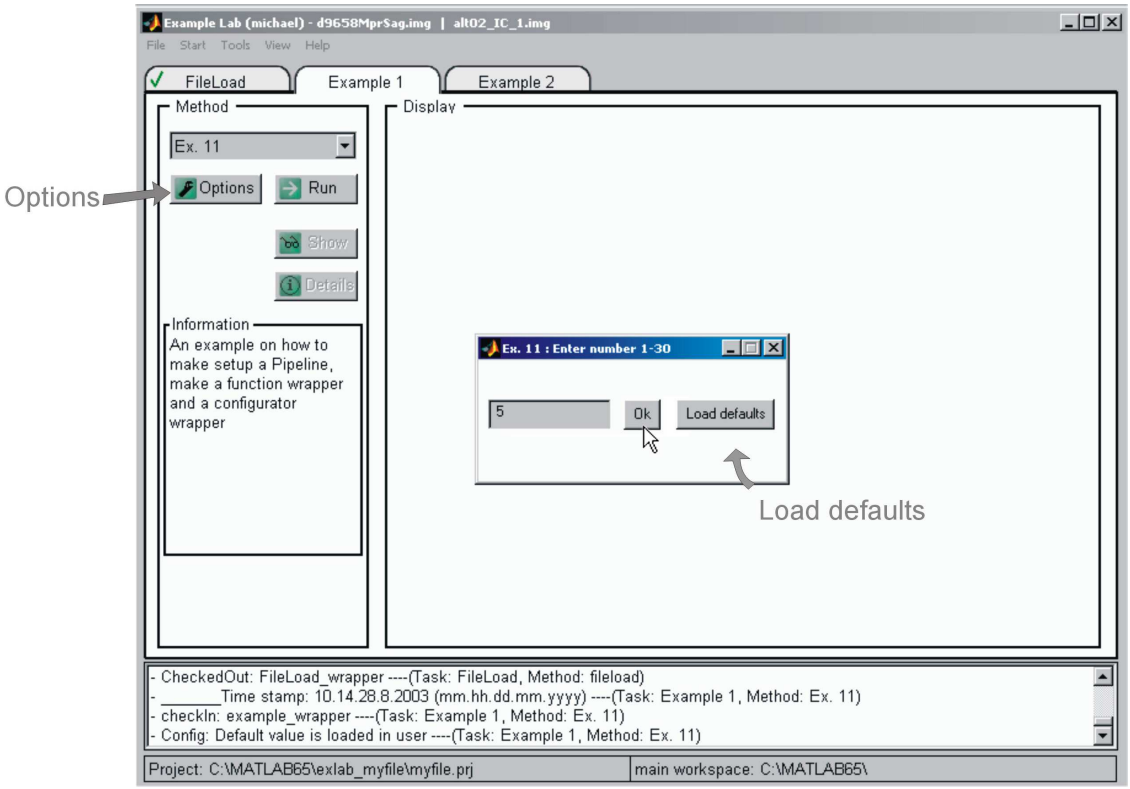

Figure 19 Pressing the 'OPTION' button is starting the configurator for the selected method Ex. 11. User can set the number iterations the method Ex. 11 should use. 'OK' button accepts the choice and 'LOAD DEFAULTS' load default values.

4.5.1 Cut and paste of 'setupExampleLab.ini'

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------- Setup file (*.ini) for The pipeline program ------
%   NAME: setupExamplelab.ini
%   SYNTAX: Matlab m-file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   EXECUTION:  0) (Done once) Ensure that The pipeline program exist in Matlab search path.
%               1) Start Matlab version 6.5 or later.
%               2) > setupPipeline('setupExamplelab.ini');
%   DESCRIPTION:
%       An example setting up a pipeline in The Pipeline Program,
%           the order of tasks and available methods for each task.
%           The setup is done by fullfill fields of at setup structure, which is loade into
%           the project initialising The Pipeline Program:
%           ('project.pipeline.taskSetup{TaskIndex,MethodIndex}').
%
% A description of the fields in the setup structure is given in the following
% (the setup structure can be copied and used):
%
% %%%%%% NOTE: ONLY GIVEN SETUP FIELDS ARE ACCEPTED BY THE PIPELINE PROGRAM %%%%%%
%_____________________________________________________________________________________
%   tmpTask=[]                          %(Matlab code) Resets the temporary data structure.
%
%   TaskIndex=                          %(numeric) Order of current task in the pipeline:
%                                       1 is the first task and placed left.
%   MethodIndex=                          %(numeric) Order of current methods in current task:
%                                       1 is first and placed highest in the popup menu.
%   tmpTask.task=                         %(string) Name of task to appear in the pipeline.
%   tmpTask.method=                         %(string) Name of method in short term to appear in popup menu.
%   tmpTask.method_name=                        %(string)(OPTIONAL) Full name of method to appear as hint
%                                       in the user-interface.
%   tmpTask.function_name=                      %(string) (OPTIONAL) Name of function (*.m, *.exe or alike)
%                                       realising the method.
%   tmpTask.description=                    %(string) Short description for the method will appear in
%                                       the information frame, when methos is selected.
%   tmpTask.require_taskindex=                  %(numeric) Which tasks (one or more) has to be done before
%                                       execution of current task.ex. [1,2].
%   tmpTask.function_wrapper=                   %(string) Name of function wrapper used to interface the pipeline
%                                       program and method.
%   tmpTask.prefix=                       %(string) Prefix given to ouputfiles if empty no change==no outputfile.
%   tmpTask.documentation=                      %(string) Where to find documentation of current method, will appear
%                                       in 'help' menu.
%   tmpTask.who_did=                        %(string) Who did the programming/development of current method, will
%                                       appear in 'help' menu.
%   tmpTask.configurator_wrapper=                 %(string)(OPTINAL) If a configurator is available this function interface
%                                       between the pipeline program and the configurator.
%   tmpTask.configurator.default=               %(string/numeric/datastructure)(OPTIONAL) Default settings of method.
%   tmpTask.configurator.user=                  %(string/numeric/datastructure) (OPTIONAL) User selected settings of method.
%   tmpTask.configurator.configurator_name=         %(string) (OPTIONAL) Configurator function (*.m,*.exe or alike)
%                                       called by the configurator_wrapper.
%   tmpTask.versionlabel=                   %(string) Textstring of used SW version number found in the functions.
%                                       Is used to track settings of a pipleine.
%
%
%   pipeline.taskSetup{TaskIndex,MethodIndex}=tmpTask;  %(Matlab code) Update settings above in the data structure
%                                       of the project 'project.pipeline.taskSetup{TaskIndex,MethodIndex}.
%________________________________________________________________________________________%_________
_______________________________________________________________________________________
%SW version: 250303TD, T. Dyrby, 120303, NRU


%__________ Image information
pipeline.imageModality={'Structural','Receptor'}; %Name and order of image modalities to be loaded
pipeline.imageIndex=1; %(FOR FURTURE USE) Number of loaded different images


%__________ Project Information
pipeline_ProjectTitleName='Example Lab';%(OPTIONAL) Title to appear in The Pipeline Program, if not given setup filename is used
pipeline_ProjectPrefix='exlab';        %(OPTIONAL) Prefix added to project files/directories to distinguish between used pipelines


%_________ Setup default pipeline
pipeline.defaultPipeline=[1,1,2];%(OPTIONAL) Default selected methods in pipeline. If empty first methods are automatically selected


%------------------------------------------------------------------------------------------------------------
%---------------- Setup the pipeline: project.pipeline.taskSetup{TaskIndex,MethodIndex} ----------------------
%------------------------------------------------------------------------------------------------------------

%______________setup FIRST TASK: FILE LOAD________________________________________________________

tmpTask=[];

TaskIndex=1;% FileLoad
```

```
MethodIndex=1;% FileLoad
tmpTask.task='FileLoad';

tmpTask.method='fileload';
tmpTask.method_name='FileLoad';
tmpTask.function_name='FileLoad_wrapper';
tmpTask.description='Load imagefiles in different formats and convert to the Analyze file format';
tmpTask.require_taskindex=0;%0=no tasks has to be done
tmpTask.function_wrapper='FileLoad_wrapper';
tmpTask.prefix='';
tmpTask.documentation='No available';
tmpTask.who_did='NRU, 2003';
tmpTask.configurator_wrapper='';
tmpTask.configurator.default='';
tmpTask.configurator.user='';
tmpTask.configurator.configurator_name='';
tmpTask.versionlabel='SW version';

pipeline.taskSetup{TaskIndex,MethodIndex}=tmpTask;


%_____setup SECOND TASK:Eaxmple 1_____________________________________________
tmpTask=[];

TaskIndex=2;% Example
MethodIndex=1;% Example 1
tmpTask.task='Example 1';

tmpTask.method='Ex. 11';
tmpTask.method_name='Example 11';
tmpTask.function_name='';                    %Is implemented in function_wrapper
tmpTask.description='An example on how to make setup a Pipeline, make a function wrapper and a configurator wrapper';
tmpTask.require_taskindex=1;
tmpTask.function_wrapper='example_wrapper';
tmpTask.prefix='XXX';
tmpTask.documentation='';
tmpTask.who_did='Tim Dyrby, NRU, 2003';
tmpTask.configurator_wrapper='exampleConfig_wrapper';
tmpTask.configurator.default='5';
tmpTask.configurator.user='';
tmpTask.configurator.configurator_name='';
tmpTask.versionlabel='SW version';
pipeline.taskSetup{TaskIndex,MethodIndex}=tmpTask;


%_____setup THIRD TASK: Example 2_____________________________________________
tmpTask=[];

TaskIndex=3;% Example
MethodIndex=1;% Example 11
tmpTask.task='Example 2';

tmpTask.method='Ex. 21';
tmpTask.method_name='Example 21';
tmpTask.function_name='';                    %Is implemented in function_wrapper
tmpTask.description='An example on how to make setup a Pipeline, make a function wrapper and a configurator wrapper';
tmpTask.require_taskindex=2;
tmpTask.function_wrapper='example_wrapper';
tmpTask.prefix='';
tmpTask.documentation='';
tmpTask.who_did='Tim Dyrby, NRU, 2003';
tmpTask.configurator_wrapper='exampleConfig_wrapper';
tmpTask.configurator.default='20';
tmpTask.configurator.user='';
tmpTask.configurator.configurator_name='exampleConfig_wrapper';
tmpTask.versionlabel='SW version';
pipeline.taskSetup{TaskIndex,MethodIndex}=tmpTask;


%_______
tmpTask=[];

TaskIndex=3;% Example
MethodIndex=2;% Example 21
tmpTask.task='Example 2';

tmpTask.method='Ex. 22';
tmpTask.method_name='Example 22';
tmpTask.function_name='';                    %Is implemented in function_wrapper
tmpTask.description='An example on how to make setup a Pipeline, make a function wrapper. NOTE: no configurator available';
tmpTask.require_taskindex=2;
tmpTask.function_wrapper='example_wrapper';
tmpTask.prefix='';
tmpTask.documentation='';
tmpTask.who_did='Tim Dyrby, NRU, 2003';
tmpTask.configurator_wrapper='';
tmpTask.configurator.default='';
tmpTask.configurator.user='';
```

```
tmpTask.configurator.configurator_name='';
tmpTask.versionlabel='SW version';
pipeline.taskSetup{TaskIndex,MethodIndex}=tmpTask;
```

4.5.2 Cut and past of 'example_wrapper.m'

```
function project=example_wrapper(project,TaskIndex,MethodIndex,varargin)
% exampel_wrapper is a demo/exampel of a method on how to implement a wrapper
%   function and read/write information into the display and log window,
%   and read information from the project structure.
%
% Inputfiles are read and saved again as filenames given in outputfiles.
%   Further progressing data is written onto the display window.
%   The example function can be used as a method for all type of tasks.
%
% Input:
%   project     : Structure containing all information of actual pipeline
%   TaskIndex   : Index to active task in the project structure
%   MethodIndex : Index to used method in a TaskIndex in the project structure
%   varargin    : Abitrary number of input arguments. (NOT USED)
%
% Output:
%   project     : Return updated project
%
% Uses special functions:
%   logProject
%   ReadAnalyzeImg
%   WriteAnalyse
%_____________________________________________
% T. Dyrby, 250803, NRU
%
%SW version: 270803TD

%_____ Show progressing data in Display window

%%_____  Show status information in the Logbar of the pipeline program
project=logProject('Example: Progress data to display window...',project,TaskIndex,MethodIndex);

%_____  Show in Display a bitmap (Start up image)(without axis)...
axes(project.handles.h_display);  %Select Display given in project
image(imread('welcome.bmp')); axis off

% Show it in 3 sec before continue with a message in the logbar...
project=logProject('Example: Startup picture for PVELab is shown in the Display in 3 sec.', project, TaskIndex,MethodIndex);
pause(3)

%_____ Show in Display some pseudo progressing data...
axes(project.handles.h_display);  %Select Display given in project
e = [];
cla
axis on
for k=1:100
   e = [e 20*exp(-k*0.4)+(max(randn(1),-1)+1)*exp(-k*0.05)];
   if(k==1)
      p=plot(pi*[1:k],e,'b','EraseMode','none');%Special plotting to avoid drawnow be slower and slower, but still update
      hold on
   else
      set(p,'XData',pi*[1:k],'YData',e)
   end
   drawnow
   axis([0 k*pi -0.5 10]);
end
hold off


%______ Initialise file load
% How many modalities exist for given bain image...
noModalities=length(project.pipeline.imageModality);% Get number of modalities e.g. PET, T1se (MR)
ImageIndex=project.pipeline.imageIndex(1);%One image alwas exist


%______ Load input files given in project and save them as output files w. prefix if prefix exists...
for(ModalityIndex=1:noModalities)
   % Get name and path of inputfiles. Note alwas use 'fullfile' to make a path independet of operation system
   % Note: Input files has already been check for existing, before entering the example_wrapper!

sourceFilename=fullfile('',project.taskDone{TaskIndex}.inputfiles{ImageIndex,ModalityIndex}.path,project.taskDone{TaskIndex}.inputfiles{ImageInd
ex,ModalityIndex}.name);

   %Get name and path of outputfiles. Note always use 'fullfile' to make a path independent of operation system
   % Note: Output files names are automatic generated and there fore READY TO USE!!!!(Very easy... ;-))

destinationFilename=fullfile('',project.taskDone{TaskIndex}.outputfiles{ImageIndex,ModalityIndex}.path,project.taskDone{TaskIndex}.outputfiles{Im
ageIndex,ModalityIndex}.name);

   % If input output filenames are equal no prefix is given
   %  it is the expected that no out put image will be available for this method
   %  this could eg. be a co-registration method where only a filename containing the
   %  co-registration matrix is found!
   %  This matrix should be stored as a field in userdata eg. 'project.taskDone{TaskIndex}.userdata.AIRfile
```

```
    if(strcmp(sourceFilename,destinationFilename))
        continue
    end

    % Show again status information in the Logbar of the pipeline program
    project=logProject('Example: Reading inputfiles',project,TaskIndex,MethodIndex);

    % Input out filenames are not equel, prefix exist
    %  Then load inputimage given in Analyze format...
    [img,hdr]=ReadAnalyzeImg(sourceFilename);

    % ...And save the image as given in the ouputfilename
    hdr.name=destinationFilename;
    project=logProject('Example: Writing to outputfiles',project,TaskIndex,MethodIndex);
    result=WriteAnalyzeImg(hdr,img);

    % Save memory...
    clear img;
end

% The wrapper is now finish at return to the pipelineprogram saying
%   that everything went well, so the next task in the pipeline can begin!!
```

4.5.3 Cut and past of ‘exampleConfig_wrapper.m’

```
function project=exampleConfig_wrapper(project,TaskIndex,MethodIndex,varargin)
% Example Config_wrapper
%
% Temporaly UserConfig is saved in the 'UserData' field in the fig. sturcture of
%   the configurator window. On exit this is handled back to project w. updated settings
%
% userConfig fields:
%       .example: an integer value 1-10
%
% Input:
%   project     : Structure containing all information of actual pipeline
%   TaskIndex   : Index to active task in the project structure
%   MethodIndex : Index to used method in a TaskIndex in the project structure
%   varargin{1} : When close GUI, 'h_Reslice=varargin{1}'
%
% Output:
%   project     : Return updated project
%_____________________________________
% T. Dyrby, 300703, NRU
%SW version: 300703TD

%_____ Set default configuration if it does not exist
if(isempty(project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.user))
   %Check if default value is given. If not set value '3'
   if(isempty(project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.default))
      userConfig.example=3;
      project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.default=userConfig;
      project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.user=userConfig;
      %loginfo
      project=logproject(('Config: No default value given is setup: NEW default=3'),project,TaskIndex,MethodIndex)
   else
      %'Default' exist in project: load into 'user' as a default value
      userConfig.example=project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.default;
      project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.user=userConfig;
      %loginfo
      project=logproject(('Config: Default value is loaded in user'),project,TaskIndex,MethodIndex)
   end
end

%____ Setup and start user-interface of the Example Configurator
h_options_example=figure('CreateFcn',{@MakeFigure_CreateFcn, project,TaskIndex,MethodIndex,varargin},'Tag','tag_options_ex');

%_____Wait for UIRESUME, continues when user presses 'ok'
uiwait(h_options_example)

%____ Get updatede project temporaly stored in the configurator fig
project=get(h_options_example,'UserData');

%____ Shout down the configurator user-interface
delete(h_options_example)

%____ Flush graphics to screen
drawnow



function project=ExitresliceConfig_callback(h,eventdata,project_task,TaskIndex,MethodIndex,varargin)
%
% Save new settings and exit configurator
%_____________________________________
% M. Twadark and T. Dyrby, 0310703, NRU
%SW version: 260803TD

% Get project loaded into figure of exampleConfig
project=eval(project_task);

% Get value in editbox
value=get(findobj('tag','tag_editButton'),'string');

%_____check that the value is numeric and in the range 1-10
if(isletter(value))
   logProject('Warning options parameters must be NUMERIC!!!',project,TaskIndex,MethodIndex);
   value='enter number';
   return
else
   % In the range 1-30?
   value=str2num(value);
   if(value>30 | value<1)
      logProject('Warning options parameters must be 1-10',project,TaskIndex,MethodIndex);
      value='enter number';
      return
   end
end
```

```
%Save value in project
userConfig.example=value;
project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.user=userConfig;

%___ Save updated project in configurator
h_options_example=findobj('tag','tag_options_ex');
set(h_options_example,'UserData',project);

%___ Resume uiwait to exit the user-interface of exampleConfig
uiresume(h_options_example)
return



function MakeFigure_CreateFcn(h_options_example, eventdata, project, TaskIndex, MethodIndex, varargin)
%(h,eventdata,project_task,TaskIndex,MethodIndex,varargin)
%
% Create figure and UI controls
% and store project temporary in the figure og the configurator
%_____________________________________
% M. Twadark and T. Dyrby, 260803, NRU
%SW version: 260803TD

%___Store temporary project in the figure of the exampel configurator
set(h_options_example,'UserData',project);

%_____ Get size of mainGUI (the userinterface of the pipeline)
figuresize=get(0,'ScreenSize');

%____Setup the user-interface
%Size of configurator GUI
color=[252/255 252/255 254/255];
windowheight=100;
windowwidth=250;
margen=10;

%setup configurator GUI with information given in project
project_task=sprintf('get(findobj("Tag","tag_options_ex"),"UserData")');%
set(h_options_example,...
   'units','pixels',...
   'position',[figuresize(3)/2-windowwidth/2 figuresize(4)/2-windowheight/2 windowwidth windowheight],...
   'MenuBar','none',...
   'NumberTitle','off',...
   'Name',[project.pipeline.taskSetup{TaskIndex,MethodIndex}.method,' : Enter number 1-30'],...
   'Color',color,...
   'NextPlot','add',...
   'tag','tag_options_ex',...
   'resize','off',...
   'CloserequestFcn',{@ExitresliceConfig_callback, project_task,TaskIndex,MethodIndex});

%Setup edit box in configurator user-interface w. 'user' value
userConfig=project.pipeline.taskSetup{TaskIndex,MethodIndex}.configurator.user;
uicontrol('parent',h_options_example,...
   'HorizontalAlignment','left',...
   'style','edit',...
   'units','pixels',...
   'position',[30 42 100 25],...
   'String',userConfig.example,...
   'Callback',{},...
   'tag','tag_editButton'...
   ); %@ExitresliceConfig_callback, project_task,TaskIndex,MethodIndex

%Setup 'Ok' button
uicontrol('parent',h_options_example,...
   'HorizontalAlignment','left',...
   'style','pushbutton',...
   'units','pixels',...
   'position',[140 42 30 25],...
   'String','Ok',...
   'Callback',{@ExitresliceConfig_callback, project_task,TaskIndex,MethodIndex},...
   'tag','tag_okButton',...
   'TooltipString','accept value'...
   );
```