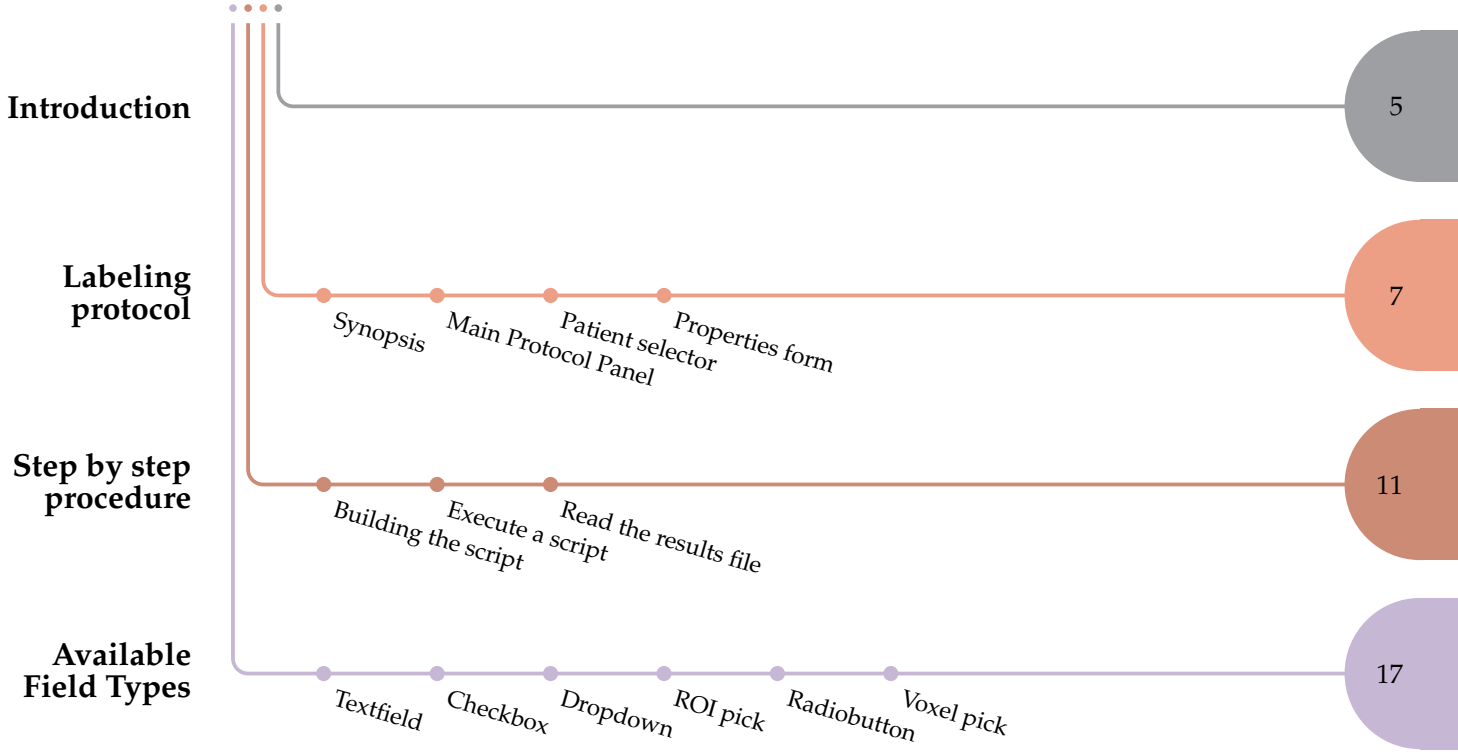
User guide

Labeling application
— LIFEx —

C. Nioche, F. Orlhac, I. Buvat

LIFEx version 6.nn,
Last update of document: 2020/08/25


Introduction
5
Labeling protocol
7
Synopsis
Main Protocol Panel
Patient selector
Properties form
Step by step procedure
11
Building the script
Execute a script
Read the results file
Available Field Types
17
Textfield
Checkbox
Dropdown
ROI pick
Radiobutton
Voxel pick

Chapter 1
Introduction

Introduction The labeling protocol allows users to easily annotate images and associated regions.

It is a generic module that makes it possible to define pre-established questions/answers for application-specific annotation and to fill in a database. This database can then be exploited for machine learning or deep learning purposes.

The questions (and possible answers) are pre-defined in a script built beforehand by the user. In the script all information necessary for the annotation task is specified. The execution of the script is then a repetition of the tasks in an automatic way with automated loading of the images and of all menus needed for the annotation.

The filling in of the annotation tasks is followed by a controller that monitors and displays the progress of the script. The user can thus know at any time the progress of his work.

The protocol consists of 3 panels that are open in the LIFEx application in three different places: 1) the controller in the master panel of the protocol (blue window at the top with the other protocols), 2) the patient selector (blue window on the left in the panel dedicated to patients), 3) the property input panel (blue window on the right next to the panel dedicated to ROI).

These 3 panels and associated actions are described in the following sections.

Chapter 2
Labeling protocol

2.1 Synopsis

This labeling protocol is composed of three different input panels (blue colors throughout the application). These panels are displayed after loading a script file. Figure 2.1, p.8. Gives an example display obtained when running this protocol.

2.2 Main Protocol Panel

Main Protocol Window. This panel displays the method to be followed for the annotation process.

Reset button. It also includes a reset button. This reset deletes the annotation result file. This deletion does NOT apply to the registered ROI files. These are not deleted and all files remain on the hard disk.

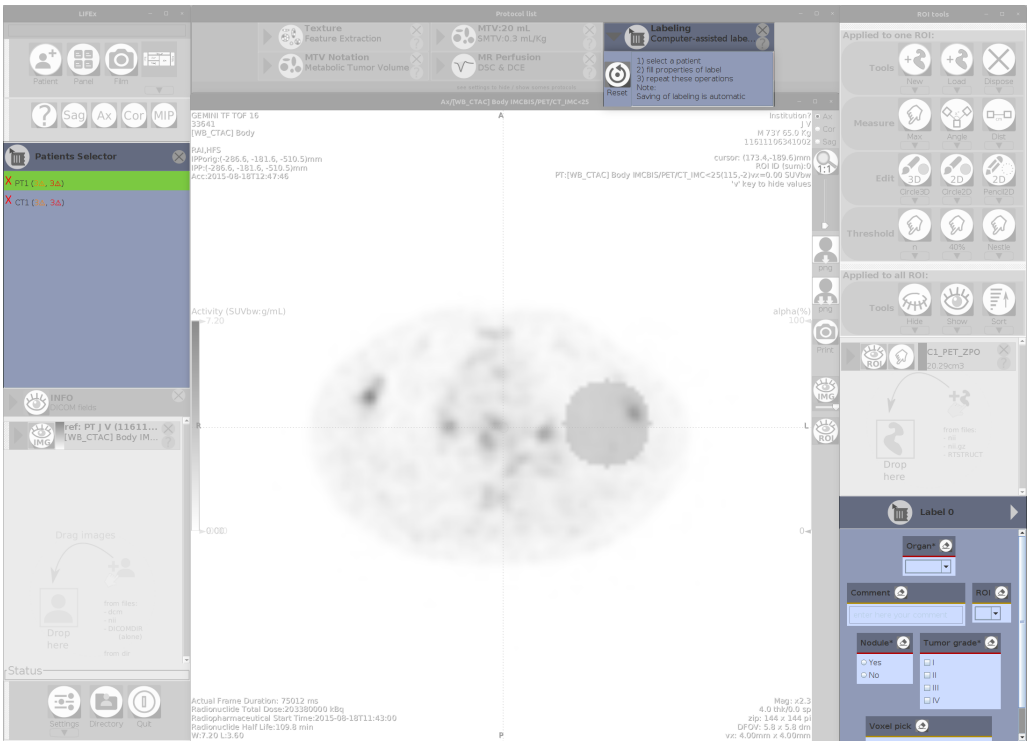


Figure 2.1: Main in GUI

2.3 Patient selector

The list of patients defined in the script are indexed and displayed in the patient selector list. These are the file names that will be retrieved and displayed for the annotation process.

This list includes additional information, such as the progress of the scoring as indicated by red crosses and green checkmarks. Each patient whose annotation is not completed is shown with a red cross (figure 2.3, p.9). All patients for whom the annotation is fully completed are shown with a green tick (figure 2.4, p.10).

To the right of the patient's name in this list are the missing data and warnings associated with the completeness of the annotation. The "warnings" are displayed in orange color and correspond to the number of fields not filled in but that are not mandatory. The "missing data" are displayed in red color and correspond to the mandatory fields that are not filled in (figure 2.3, p.9).

2.4 Properties form

2.4.1 Graphic user interface of properties

Form. The property entry form is built from the definitions given in the script. Each definition will be associated with a property and thus a dialog box.

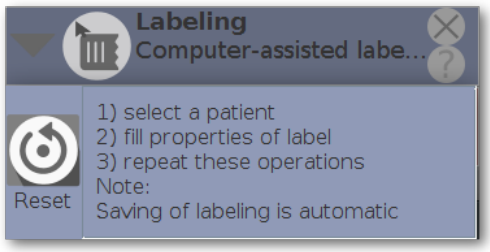


Figure 2.2: ProtocolGui

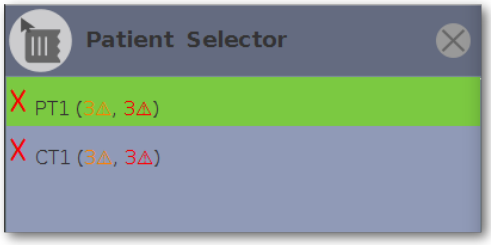


Figure 2.3: Patient Selector with showing the patients that still need to be annotated

Label. The properties are grouped under a label. The user can add as many labels as he wants. Navigation buttons (right and left arrows) are set on both sides of the label title. This will display the 1st label "label 0", then the 2nd label "label 1" and so on.

Note. Each property associated with a label will have to be filled in to validate the complete annotation of the patient (green tick displayed).

2.4.2 Three audit levels = three color levels

The annotation form includes 3 visual levels of field verification.

1. Mandatory field with missing data (red line): The field is mandatory (title with *) and is still empty;
2. Field with warning (orange line): The warning indicated that the field is not mandatory but is empty;
3. Field ok (green line): The field is properly filled in.

Figure 2.4: Patient Selector showing that all patients have been annotated

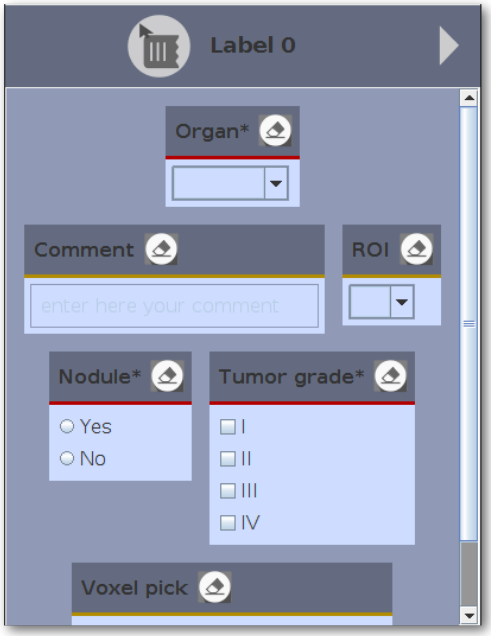


Figure 2.5: Properties Gui

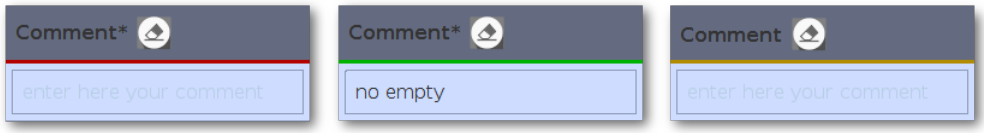


Figure 2.6: Example of missing annotation (red line), field properly filled in (green line), field requiring attention (orange line)

Chapter 3
Step by step procedure

3.1 Building the script

What is a script? The script file is a text file that can be easily read and edited. It has the following file syntax: key=value.

In this line: `script=labeling`
→ the key is 'script' and its value is 'labeling'. The set of properties that makes up the script is written using the same syntax.

Understanding the different sections of the script. The script is composed of several sections. The first `Common` section is fixed and must not be altered. It informs the application about the type of script that follows.

3.1.1 Fixed and common sections

`script = labeling`
→ Script is for labeling

`script.version = 1.0`
→ version of this script is 1.0

Step by step procedure

The common and fixed sections of the script file are to be copied as is. Please, do not modify them. It makes it possible for the application to understand the rest of the script.

3.1.2 Result section

`output.props.file=.../LabelingResults.props`
→ this key indicates the output file in which the annotation results will be stored. The path and filename must respect the syntax of the system on which you are running the application. In Windows system the syntax C:/directory/home should be used (the \ are translated into /).

3.1.3 Properties section

`'#'` indicates the property number, it should start from 0, and should be incremented by 1 each time a new property is added.

`property#.title`
→ is the property name (add ∗ when an answer is absolutely required for that property). Example for an mandatory answer to a Tumor location question, set `property0.title=Tumor location*` for property0.

`property#.title.tooltiptext`
→ optional, is a tool tip text added to the property name

`property#.type`
→ mandatory [radiobutton | checkbox | textfield | dropdown | voxelPick | roiPick]. Example for checkbox of property0: `property0.type=checkbox`. See chapter Available Field Types 4 p.17 for more explanation regarding the different field type.

`property#.value`
→ mandatory, is the list of property values to be selected from, several values are available using "|" as a character separator. Example of I,II,III,IV tumor grade for property0: `property0.value=I|II|III|IV`.

`property#.value.tooltiptext`
→ optional, is a tool tip text added to value property, several texts can be included using "|" as a character separator. Example of `property0.value.tooltiptext=Select this checkbox for I tumor grade|Select this checkbox for II tumor grade|Select this checkbox for III tumor grade|Select this checkbox for IV tumor grade`

3.1.4 Patient/Image section

From nifti files:
`patient#.img#=.../file`
→ mandatory, is the path to the image file to be read. It may be a nifti file. For instance, `patient0.img0=/home/user/I1.nii` to select the first patient (patient0) and read image I1 (img0) for that patient from the /home/user directory.

From directory with dicom files:
`patient#.img#=.../directory`
→ mandatory, is the path to the image file to be read. It may be a directory that contains files. For instance: `patient0.img0=/home/user/P1` to select the first patient (patient0) and read the first image (img0) in the directory P1 of /home/user.

From DICOMDIR file:
`patient#.img#=.../DICOMDIR`
→ mandatory, is the DICOM file.

3.1.5 Optional section

```
# add get button on all voxelpick properties -> optional [ true | false ],
default:  true properties.voxelpick.getLocalisationOnViews = false
```
→ this true/false property allows you to add (or not) an additional button to locate the coordinates entered in this field on the slice plan. When this button is pressed, the display cross is placed on these coordinates and the views are updated.

```
# directory of ROI files -> optional
output.roi.file=.../roi
```
→ is the directory where ROI files will be saved. This is optional because if it is not defined, the directory set in the application will be reused (see `Settings\ FrameGui\ DataDir`). The path and filename must respect the syntax of the system on which you are running the application. With Windows system, the syntax C:/directory/home should be used for which the \ are translated into /.

```
# is property heading saved (in addition to the values) -> optional
# [ true | false ], default:  false
output.title.saved=false
```
→ You can set this key to true if you want the name of the property to be included in the results file. This can help human proofreading. In this case, the read property of the script will be rewritten in the output file. By default, if the property is not filled in, it will be assigned to 'false'.

```
# Window Leveling -> optional
#all.patient.img0.window.width=10
#all.patient.img0.window.level=5
```
→ It is possible to voluntarily set the windows/Leveling at certain values. To do this, simply fill in these values in the corresponding fields

```
# color map ; available options
```

```
# MONOCHROME1 || MONOCHROME1_PVALUES || MONOCHROME1_PVALUES0
# MONOCHROME2 || MONOCHROME2_PVALUES || MONOCHROME2_PVALUES0
# RAINBOW || RAINBOW_PVALUES || RAINBOW_PVALUES0
# RAINBOW_INVERSE || RAINBOW_INVERSE_PVALUES || RAINBOW_INVERSE_PVALUES0
# RAINBOW_NEW || RAINBOW_NEW_PVALUES || RAINBOW_NEW_PVALUES0
# RAINBOW_NEW_INVERSE || RAINBOW_NEW_INVERSE_PVALUES
# || RAINBOW_NEW_INVERSE_PVALUES0
# HEAT || HEAT_PVALUES || HEAT_PVALUES0
# HEAT_INVERSE || HEAT_INVERSE_PVALUES || HEAT_INVERSE_PVALUES0

# color map in first 3 frames -> optional, default:  MONOCHROME2
frame0.img.color=MONOCHROME2
frame1.img.color=MONOCHROME2
frame2.img.color=MONOCHROME2
```

→ It is possible to set the color palettes on playback according to each frame (frame0, frame1 & frame2). The name of the color palette must be defined. The possible choices are listed above.

3.1.6 Full example

```
# Common
# fixed information on script -> mandatory
script = labeling
script.version = 1.0

# result file -> mandatory
output.props.file=.../LabelingResults.props

# directory of ROI files -> optional
output.roi.file=.../roi

# is heading saved (in addition to the values) -> optional
# [ true | false ], default:  false
output.title.saved=false

# Window Leveling -> optional
#all.patient.img0.window.width=10
#all.patient.img0.window.level=5
# color map ; available options
# MONOCHROME1 || MONOCHROME1_PVALUES || MONOCHROME1_PVALUES0
# MONOCHROME2 || MONOCHROME2_PVALUES || MONOCHROME2_PVALUES0
# RAINBOW || RAINBOW_PVALUES || RAINBOW_PVALUES0
# RAINBOW_INVERSE || RAINBOW_INVERSE_PVALUES || RAINBOW_INVERSE_PVALUES0
# RAINBOW_NEW || RAINBOW_NEW_PVALUES || RAINBOW_NEW_PVALUES0
# RAINBOW_NEW_INVERSE || RAINBOW_NEW_INVERSE_PVALUES
# || RAINBOW_NEW_INVERSE_PVALUES0
# HEAT || HEAT_PVALUES || HEAT_PVALUES0
# HEAT_INVERSE || HEAT_INVERSE_PVALUES || HEAT_INVERSE_PVALUES0

# color map in first 3 frames -> optional, default:  MONOCHROME2
```

```
frame0.img.color=MONOCHROME2
frame1.img.color=MONOCHROME2
frame2.img.color=MONOCHROME2


# properties
#
# # Number of properties
# must be incremented by 1, start from 0

# property#.title -> is heading of property (add * when an answer is absolutely
required)
# property#.title.tooltiptext -> optional, is tool tip text added to heading
property
# property#.type -> mandatory [ radiobutton | checkbox | textfield | dropdown
| voxelpick | roipick ]
# property#.value -> mandatory, is property values available to be selected
from, several values are available with "|" character separator
# property#.value.tooltiptext -> optional, is tool tip text added to value
property, several texts are available with "|" character separator

# see examples, below:
# textfield
property0.title = Organ*
property0.type = dropdown
property0.value = Lung|Kidney|Stomach|Liver

# comment
property1.title = Comment*
property1.type = textfield

# roipick
property2.title = ROI
property2.type = roipick

# radiobutton
property3.title = Nodule*
property3.title.tooltiptext = This is the tool tip text corresponding to
the nodule heading
property3.type = radiobutton
property3.value = Yes|No
property3.value.tooltiptext = presence of a nodule|no nodule

# checkbox
property4.title = Tumor grade*
property4.type = checkbox
property4.value = I|II|III|IV

# voxelpick
property5.title = Voxel pick
property5.type = voxelpick
```

```
#
# image of patients or directory
patient0.img0=.../img/I0
patient1.img0=.../img/I1
patient2.img0=.../img/I2
```

Step by step procedure

3.2 Execute a script

The script is executed by dragging and dropping the file containing the script in the patient loading area on the left side of the application. The application then starts executing automatically.

3.3 Read the results file

The results are stored in the file specified in the script using the key 'output.props.file'. This file has the same syntax as that of the script, i.e. a key=value pair repeated on each line.

Example of results file:

```
#LIFEx5.82 Labeling properties
#Thu Apr 16 19:40:08 CEST 2020
patient0.label0.property0.value=Stomach
patient0.label0.property1.value=comment
patient0.label0.property2.value=
patient0.label0.property3.value=Yes
patient0.label0.property4.value=III
patient0.label0.property5.value=t0 z0 y61 x62
patient1.label0.property0.value=Lung
patient1.label0.property1.value=another comment
patient1.label0.property2.value=
patient1.label0.property3.value=No
patient1.label0.property4.value=IV
patient1.label0.property5.value=t0 z0 y66 x23
```

Explanation of the example file. This result file contains the annotations for two patients (patient0 & patient1). Each patient has only one defined label (label0), and the values of the 6 properties (property0...5) are recorded without their headings.

The opening of this file in an excel file is possible, by giving the symbol '=' (with the optional additional symbol '.') as delimiter.

Chapter 4
Available Field Types

Several types of fields are available to define a complete form. These fields are taken from the main generic fields of programming languages and are: Textfield, Checkbox, Dropdown, Radiobutton, Voxel pick and ROI Dropdown. All are described below.

4.1 Textfield

The TextField (`property#.type=textfield`) component is a complete form control including a label, input and help text. Text fields let users enter and edit text (fig 4.1 p.18).

4.2 Checkbox

The checkbox (`property#.type=checkbox`) is shown as a square box that is ticked (checked) when activated. Checkboxes are used to let a user select one or more options

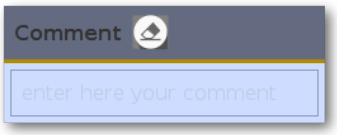


Figure 4.1: textfield type

among a limited number of choices (fig 4.2 p.18).

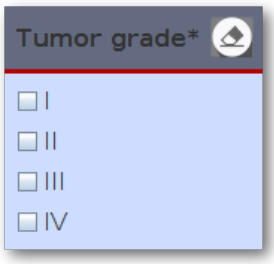


Figure 4.2: checkbox type

4.3 Dropdown

Dropdown (`property#.type=dropdown`) is a list of choices appearing below a menu title when it is selected, and remaining until used or dismissed (fig 4.3 p.18).

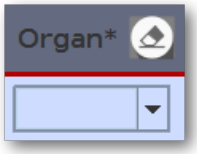


Figure 4.3: dropdown type

4.4 ROI pick

Dropdown for ROI (`property#.type=roipick`) is a list of ROI that is hidden until you choose to look at it (fig 4.4 p.19).

4.5 Radiobutton

Radio buttons (`property#.type=radiobutton`) are normally presented in radio groups (a collection of radio buttons describing a set of related options). Only one radio button in a group can be selected at the same time (fig 4.5 p.19).

Figure 4.4: roipick type

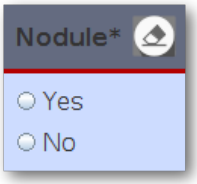


Figure 4.5: radiobutton type

4.6 Voxel pick

The TextField (`property#.type=voxelpick`) component is a complete form control including a label, input and help text. Text fields let users enter coordinate voxel only by mouse click (fig 4.6 p.19).

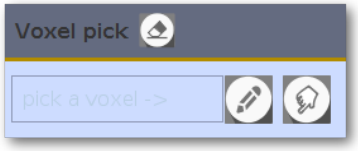


Figure 4.6: voxelpick type