

Scripts

Local Image Features **Extraction** — **LIFEx** —

C. Nioche, F. Orhac, I. Buvat

LIFEx version 26.1.n,
Last update of document: 2026/02/02

Part I

What is a script?

page 7

Introduction

Rationale and what is a script?
General information
Script execution sequence
How to run a script file
Multi-scripts execution

9

Part II ...

General concepts

page 13

How to create?

Write a script
Script for getting started
More complex script
Main remarks
Perform operations
Perform common setting

15

Common properties

US Properties

19

Series operations

Syntax examples
Script auto-completion with patient tree (v>=25.10.1)
Series common operations
Series operations

21

ROI operations

Syntax examples
ROI operations from threshold menu
ROI operations from file/edit menu
ROI operations from measure menu
ROI operations from sort menu

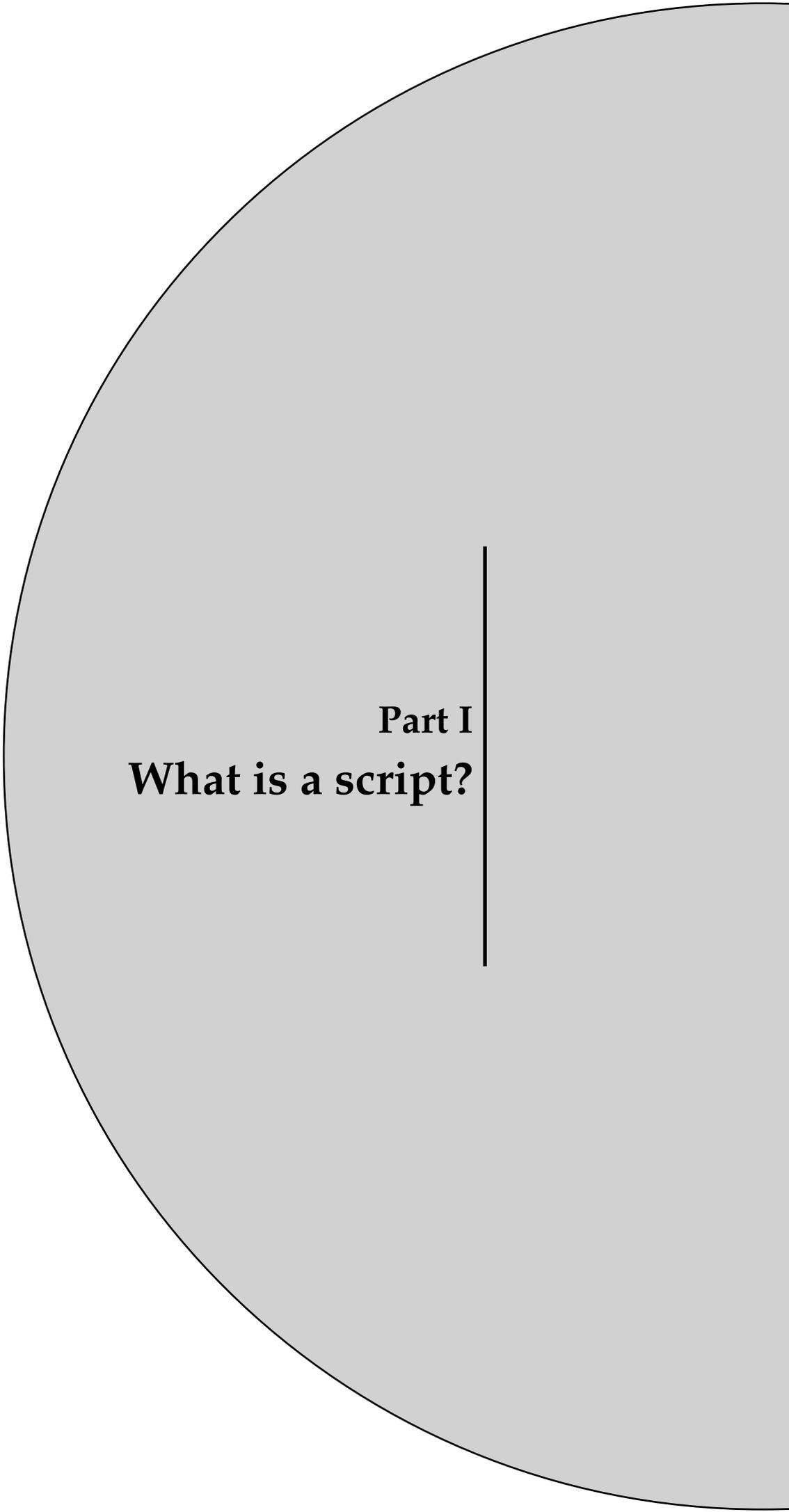
33

Protocol operations

Protocol TEXTURE operations
MTV Protocol operations

37

List of Figures



Part I
What is a script?

Chapter 1

Introduction

1.1 Rationale and what is a script?

Scripting consists of developing instructions that automate processes in the LIFEx application. It is commonly employed to streamline repetitive tasks, perform complex computations, or adapt the user experience for research.

In practice, a script enables execution of all operations and calculations without human intervention. Multiple scripts may be prepared, arbitrarily named, stored, and modified as required, as they are plain text files.

The LIFEx scripting procedure generally consists of the following steps:

- **Script writing:** involves employing the syntax and commands of the scripting language to define executable instructions for the application.
- **Script debugging and testing** consist in executing the script, validating the output, and correcting any errors to ensure conformity with expected behavior.

1.2 General information

- Integration and execution: once functional, the script must be executed within the application, with explicit specifications for timing and mode of execution.
- Script maintenance: ongoing adaptation is required to ensure functionality with evolving application versions, including bug corrections and potential feature extensions.

1.2 General information about writing a script text file

Introduction: A LIFEx script consists of a plain text file structured as a sequence of properties, each defined on a separate line.

Properties: Properties are configuration values expressed as key=value pairs. The key serves as an identifier analogous to a variable, used to access the associated value.

Property example: Example: an application may use the property "*LIFEx.Output.Directory*" to specify the directory in which files are stored.

LIFEx.Output.Directory = /home/user/newDirectory

- key property is "*LIFEx.Output.Directory*" Upper and lower case are not important in the key. This key can also be written : "*LIFEx.output.directory*"
- value property is "*/home/user/newDirectory*"

Construction guidelines of property:

- warning: Keys must be unique identifiers, with a single definition per line.
- Leading spaces preceding a key are disregarded.
- The order of property lines is functionally irrelevant; however, categorization enhances readability for users.
- File paths must employ the / separator consistently, independent of the operating system.
- Comment lines, indicated by the character #, are excluded from application parsing.

1.3 Script execution sequence

The execution of scripts follows a mandatory sequence, as outlined below:

1. Series loading [mandatory]
2. — Series operations [optional]
3. — Series saving [optional]
4. ROI loading [optional]
5. — ROI operations [mandatory when ROI is loaded, otherwise optional]

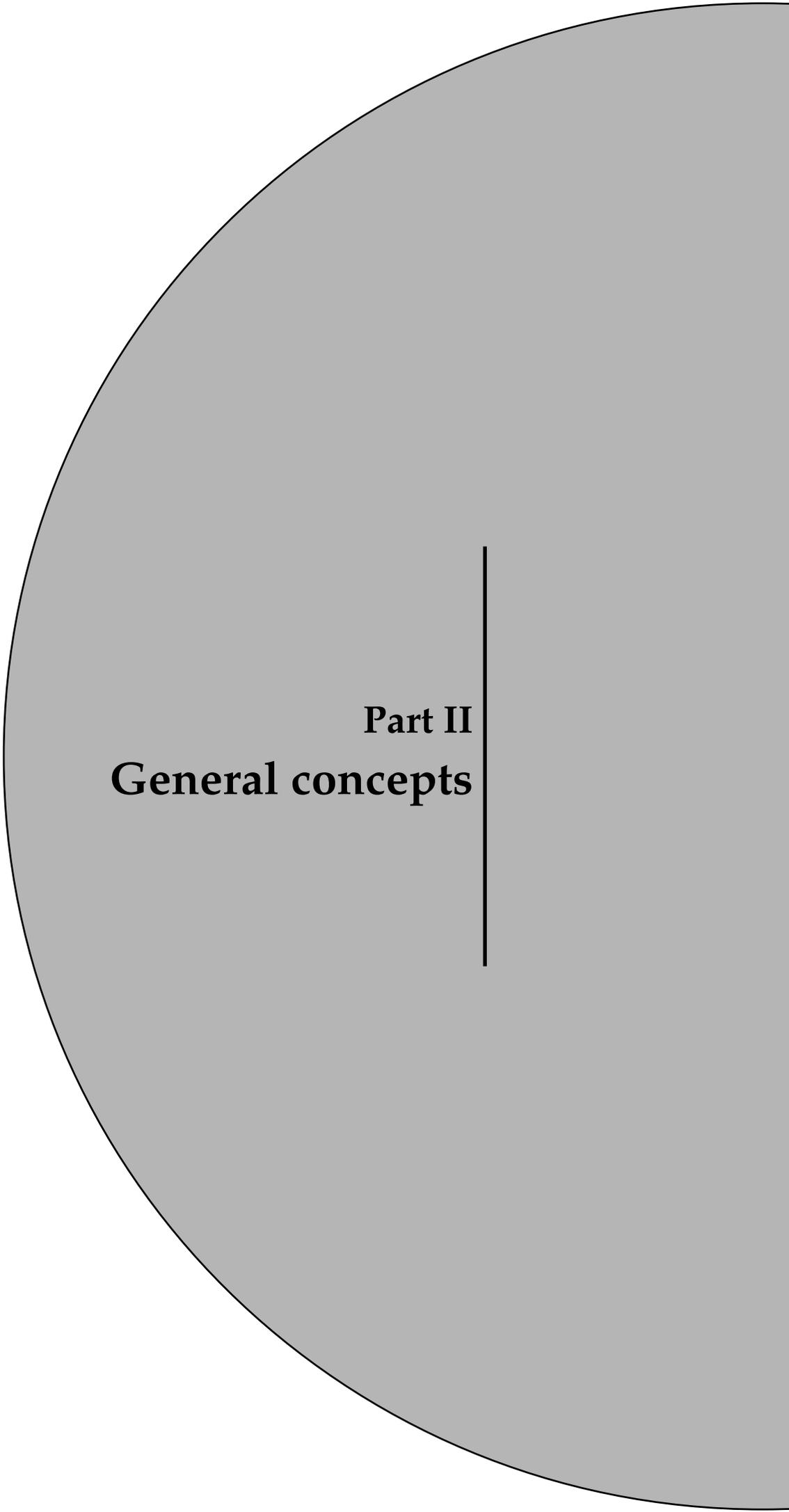
6. — ROI saving [mandatory when ROI is loaded, otherwise optional]
7. Protocol loading [optional]
8. — MTV protocol operations [optional]
9. — Texture protocol operations [optional]
10. — ROI closing [optional and automatic]
11. ROI closing [optional and automatic]
12. Series closing [automatic]

1.4 How to run a script file

To execute a script, the file must be dragged into the patient image loading panel. Recognition as a valid script requires its size not to exceed 2 MB.

1.5 Can I run several script files at the same time?

Multiple script files may be executed sequentially and automatically by selecting them simultaneously and dragging them into the patient image loading panel.



Part II
General concepts

Chapter 1

How to create a script?

1.1 Write a script

Scripts allow chaining image and ROI loading to produce a consolidated results file in .csv format.

1.2 Script for getting started

Example: the script loads two series (PT0, PT1), each with two associated ROIs (Patient 0 and Patient 1).

The script may be copied into a text file (e.g., script.txt). File paths must be modified to reflect the actual directories: ex. `directory/subDirectory`

1.3 More complex script

How to create?

```
#  
#  
# Common  
#  
# result directory -> mandatory  
LIFEx.Output.Directory={directory/subDirectory}  
#  
#  
# Patient 0 / Series 0 / ROI 0 / ROI 1  
#  
# loading series -> mandatory  
LIFEx.Patient0.Series0={directory/subDirectory}/PT0  
LIFEx.Patient0.Roi0={directory/subDirectory}/RoiVolume/S0R0.uint16.nii.gz  
LIFEx.Patient0.Roi1={directory/subDirectory}/RoiVolume/S0R1.uint16.nii.gz  
#  
#  
# Patient 1 / Series 0 / ROI 0 / ROI1  
#  
# loading series -> mandatory  
LIFEx.Patient1.Series0={directory/subDirectory}/PT1  
LIFEx.Patient1.Roi0={directory/subDirectory}/RoiVolume/S1R0.uint16.nii.gz  
LIFEx.Patient1.Roi1={directory/subDirectory}/RoiVolume/S1R1.uint16.nii.gz
```

1.3 More complex script: save anonymous DICOM Series from DICOM Series

The following provides a complete example of saving an anonymized DICOM series from an input DICOM series:

```
#  
#  
# Patient 0 / Series 0  
#  
# loading series -> mandatory  
LIFEx.Patient0.Series0={directory/subDirectory}/PT0  
LIFEx.Patient0.Series0.Operation0=Save anonymous series dcm  
#  
#  
# Patient 1 / Series 0  
#  
# loading series -> mandatory  
LIFEx.Patient1.Series0={directory/subDirectory}/PT1  
LIFEx.Patient1.Series0.Operation0=Save anonymous series dcm
```

1.4 Main remarks relevant to the script writing

- What are the image formats that can be managed using LIFEx scripts?
 - Image files may be provided in NIfTI-1 format (.nii or .nii.gz), requiring full path specification in the script;
 - For DICOM images, the root directory (excluding filenames) must be specified. All files within the directory are then loaded.
- What are the ROI formats that can be managed using LIFEx scripts?
 - ROI formats supported include NIfTI-1 and RTStruct. In both, the full file path must be provided; for RTStruct, all ROIs are imported without exception.
- Syntax of all pathways of files in LIFEx scripts:
 - File paths must exclude accents and spaces.
 - Path syntax is not strictly dependent on the operating system.
 - * Windows paths follow Unit:/Directory/File.extension or Unit:/Your Directory/ for series of DICOM images example C:/Home/Users1/File1
 - * Linux paths: /Your Directory/Your File.extension
 - * macOS paths: /Your Directory/Your File.extension

How to create?

1.5 Perform operations

Intermediate ROI operations may be executed after Series and ROI loading but prior to feature extraction and results recording:

Main syntax of one operation: The operation syntax is defined as:
 {key}.Operation0=nameOp0,arg1,arg2,...,argn

with "nameOp0" is the title of the button of the ROI action to be performed, arg1 the first argument, arg2 the second argument, ...

Main syntax of many operations to the same Series or ROI: so each uses the result of the previous.

{key}.Operation0=nameOp0,arg1,arg2,...,argn|nameOp1,arg1,arg2,...,argn

with "nameOp0" is the first operation, "nameOp1" is the second operation.

Operations are executed in the order in which they are written (left to right).

Here's an example:

{key}.Operation0=SpatialResampling,2,2,2|Texture,false,false,false,1,3D,Absolute,0.314,192,0,60

1.6 Perform common setting

Explanation:

- series 0 is selected to start this operation 0;
- resample rescaling is realized on series 0
- texture protocol is realized on resample rescaling series result.

Order operations between lines: If you have more than one operation to perform you can describe the actions with an order given by the key_operationN:

```
{key}.Operation0=...  
{key}.Operation1=...  
{key}.Operation2=...
```

How to create?

1.6 Perform common setting

LIFEx.Password: enables access to experimental or unreleased functions, facilitating controlled feature release.

```
LIFEx.Password=****
```

LIFEx.GuiUpdate: controls whether the GUI is displayed. Disabling it (false) can accelerate script execution, particularly in server environments (LIFEx>=7.4.1) [true (default) || false]

```
LIFEx.GuiUpdate=false||true
```

LIFEx.SeriesRoiGuiUpdate: defines whether series and ROI images are rendered during script execution. Disabling (false) accelerates computation (LIFEx>=7.5.6) [true (default) || false]

```
LIFEx.SeriesRoiGuiUpdate=false||true
```

LIFEx.MessagesGuiUpdate: determines whether textual messages are displayed. Setting to false suppresses output and increases performance (LIFEx>=7.5.6) [true (default) || false]

```
LIFEx.MessagesGuiUpdate=false||true
```

Chapter 2

Common properties

2.1 US Properties

add US Properties:

For ultrasound DICOM data, disabling the red, green, and blue channels enables acquisition of the averaged signal alone (all channels are active by default).

The following properties may be added to deactivate individual RGB channels:

```
LIFEx.PropertiesSeriesUS.haveRedChannel=false  
LIFEx.PropertiesSeriesUS.haveGreenChannel=false  
LIFEx.PropertiesSeriesUS.haveBlueChannel=false
```


Chapter 3

Series operations

3.1 Syntax examples

- Store series in anonymized DICOM format
LIFEx.Patient0.Series0.Operation0=Save anonymous dcm
- Standardize series units to SUVbw:
LIFEx.Patient0.Series0.Operation0=SUVbw
- Apply 2x2x2 resampling prior to saving in NiftI format:
LIFEx.Patient0.Series0.Operation0=SpatialResampling,2,2,2|save nii uint16
- Export results in anonymized NiftI format:
LIFEx.Patient0.Series0.Operation0=Save Anonymous nii uint16
- Define the output directory for series export:
LIFEx.Output.Directory={directory/subDirectory/}

3.2 Script auto-completion with patient tree (v>=25.10.1)

3.2.1 Script auto-completion with one series (v>=25.10.1)

It is possible to let LIFEx automatically complete the file tree (Series and ROIs), as well as the operations (on Series and ROIs). LIFEx will generate a basic script, which must include at least the following keys:

- LIFEx.Input.Directory={directory/subDirectory/}
- LIFEx.Series0.Operation0={Series operation}
- LIFEx.Roi0.Operation0={ROI operation}

Note that the "PatientN" level has disappeared, since it will be recreated during this procedure.

Neither file names nor directory names matter. The application automatically identifies each file as either an image (Series) or a segmentation (ROI).

The directory structure does not need to follow a strict format. Automatic completion will work as long as the application can detect images and ROIs in the sub-tree. Please avoid placing multiple image series in the same patient folder, and ensure that each ROI is located close to its corresponding image.

In this case, LIFEx will start by scanning the "Input Directory" and all its subfolders in order to build a detailed script, including the construction of patient0, patient1, ..., patientN. This new script file will be saved with an additional extension, so as not to overwrite the original script file.

Automatic construction has some limitations:

- only one series (series0) can be included
- only one operation can be applied to Series0
- only one operation can be applied to each ROI

Example of basic script

```
LIFEx.Input.Directory=/home/example/input
```

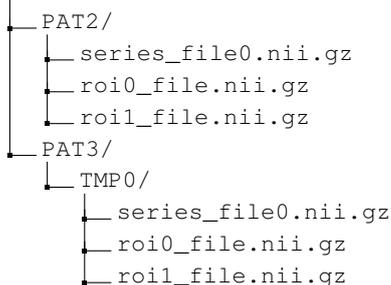
```
LIFEx.Series0.Operation0=spatialresampling,2,2,2|Texture,false,false,false,1,3D,Absolute,0.3125,192,0,60
```

```
LIFEx.Input.Directory=/home/example/output
```

File tree example (/home/example):

```
/home/example/  
├── PAT0/  
│   ├── SERIES  
│   │   └── series_file0.nii.gz  
│   └── ROI  
│       ├── roi0_file.nii.gz  
│       └── roi1_file.nii.gz  
└── PAT1/  
    ├── SERIES  
    │   ├── series_file0.dcm  
    │   ├── series_file1.dcm  
    │   └── series_file2.dcm  
    └── ROI  
        ├── roi0_file.nii.gz  
        └── roi1_file.nii.gz
```

3.2 Script auto-completion with patient tree (v>=25.10.1)



Auto-completion will write a new content script as follows:

```
lifex.patient0.roi0=/home/example/PAT0/ROI/
lifex.patient0.series0=/home/example/PAT0/SERIES/series_file0.nii.gz
lifex.patient0.series0.operation0=spatialresampling,2,2,2|Texture,false,false,false,1,3D,Absolute,0.3125,192,0,60

lifex.patient1.roi0=/home/example/PAT1/ROI/
lifex.patient1.series0=/home/example/PAT1/SERIES/
lifex.patient1.series0.operation0=spatialresampling,2,2,2|Texture,false,false,false,1,3D,Absolute,0.3125,192,0,60

lifex.patient2.roi0=/home/example/PAT2/roi0_file.nii.gz
lifex.patient2.roi1=/home/example/PAT2/roi1_file.nii.gz
lifex.patient2.series0=/home/example/PAT2/series_file0.nii.gz
lifex.patient2.series0.operation0=spatialresampling,2,2,2|Texture,false,false,false,1,3D,Absolute,0.3125,192,0,60

lifex.patient3.roi0=/home/example/PAT3/TMP0/roi0_file.nii.gz
lifex.patient3.roi1=/home/example/PAT3/TMP0/roi1_file.nii.gz
lifex.patient3.series0=/home/example/PAT3/TMP0/series_file0.nii.gz
lifex.patient3.series0.operation0=spatialresampling,2,2,2|Texture,false,false,false,1,3D,Absolute,0.3125,192,0,60
```

Series operations

Windows: On Windows, the ':' character may appear escaped with a backslash (\:). This does not affect the script when it is read back. Otherwise, all \ characters are properly converted to /

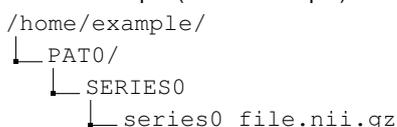
3.2.2 Script auto-completion with several series (v>=25.10.3)

When working with multiple series, the script can specify which reference series should be used, determining the loading order of the series. It is also possible to deselect one series and select another to run the same script on different datasets. Here are some examples to explain this functionality.

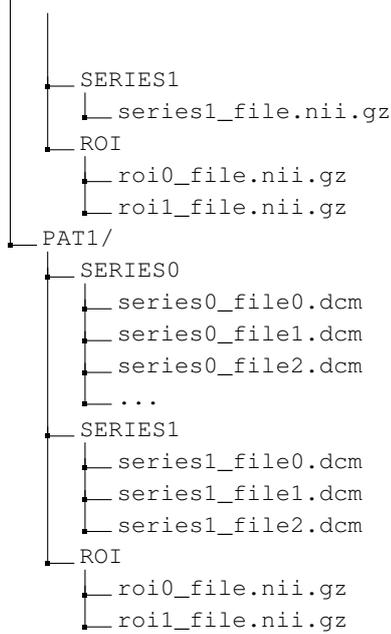
• Example 1

Two patients, each with two series and two ROIs. In this case, both series are loaded simultaneously, and one serves as the reference (based on the alphabetical order of the directories). This may not always be the intended behavior (see Examples 2 and 3).

File tree example (/home/example):



3.2 Script auto-completion with patient tree (v>=25.10.1)



Original script as follows:

```
LIFEx.Input.Directory=/home/example/
LIFEx.Output.Directory=/home/example/results

# *****
LIFEx.Series0.RelativeDirectory=SERIES0
LIFEx.Series0.Selected=true

LIFEx.Series0.Operation0=Series0Operation0Content
LIFEx.Series0.Operation1=Series0Operation1Content
LIFEx.Series0.Operation2=Series0Operation2Content

LIFEx.Roi0.Operation0=Roi0Operation0Content
LIFEx.Roi0.Operation1=Roi0Operation1Content
LIFEx.Roi0.Operation2=Roi0Operation2Content

# *****
LIFEx.Series1.RelativeDirectory=SERIES1
LIFEx.Series1.Selected=true

LIFEx.Series1.Operation0=Series1Operation0Content
LIFEx.Series1.Operation1=Series1Operation1Content
LIFEx.Series1.Operation2=Series1Operation2Content

LIFEx.Roi1.Operation0=Roi1Operation0Content
LIFEx.Roi1.Operation1=Roi1Operation1Content
LIFEx.Roi1.Operation2=Roi1Operation2Content
```

Auto-completion will write a new content script as follows:

```
LIFEx.Input.Directory=/home/example/
```

3.2 Script auto-completion with patient tree (v>=25.10.1)

LIFEx.Output.Directory=/home/example/results

```
# *****
LIFEx.Patient0.Series0.Operation0=Series0Operation0Content
LIFEx.Patient0.Series0.Operation1=Series0Operation1Content
LIFEx.Patient0.Series0.Operation2=Series0Operation2Content
LIFEx.Patient0.Series1.Operation0=Series1Operation0Content
LIFEx.Patient0.Series1.Operation1=Series1Operation1Content
LIFEx.Patient0.Series1.Operation2=Series1Operation2Content
LIFEx.Patient0.Roi0.Operation0=Roi0Operation0Content
LIFEx.Patient0.Roi0.Operation1=Roi0Operation1Content
LIFEx.Patient0.Roi0.Operation2=Roi0Operation2Content
LIFEx.Patient0.Roi1.Operation0=Roi1Operation0Content
LIFEx.Patient0.Roi1.Operation1=Roi1Operation1Content
LIFEx.Patient0.Roi1.Operation2=Roi1Operation2Content
LIFEx.Patient0.Roi0=PAT0/ROI/roi0_file.nii.gz
LIFEx.Patient0.Roi1=PAT0/ROI/roi1_file.nii.gz
LIFEx.Patient0.Series0=PAT0/SERIES0/series0_file.nii.gz
LIFEx.Patient0.Series1=PAT0/SERIES1/series1_file.nii.gz

# *****
LIFEx.Patient1.Series0.Operation0=Series0Operation0Content
LIFEx.Patient1.Series0.Operation1=Series0Operation1Content
LIFEx.Patient1.Series0.Operation2=Series0Operation2Content
LIFEx.Patient1.Series1.Operation0=Series1Operation0Content
LIFEx.Patient1.Series1.Operation1=Series1Operation1Content
LIFEx.Patient1.Series1.Operation2=Series1Operation2Content
LIFEx.Patient1.Roi0.Operation0=Roi0Operation0Content
LIFEx.Patient1.Roi0.Operation1=Roi0Operation1Content
LIFEx.Patient1.Roi0.Operation2=Roi0Operation2Content
LIFEx.Patient1.Roi1.Operation0=Roi1Operation0Content
LIFEx.Patient1.Roi1.Operation1=Roi1Operation1Content
LIFEx.Patient1.Roi1.Operation2=Roi1Operation2Content
LIFEx.Patient1.Roi0=PAT1/ROI/roi0_file.nii.gz
LIFEx.Patient1.Roi1=PAT1/ROI/roi1_file.nii.gz
LIFEx.Patient1.Series0=PAT1/SERIES0/
LIFEx.Patient1.Series1=PAT1/SERIES1/
```

Series operations

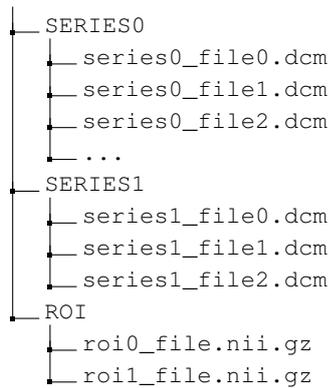
• Example 2

Two patients, each with two series and two ROIs. Note that in this case, only one of the two series will be loaded - this introduces the concept of a selected series.

File tree example (Tree structure identical to that of Example 1) (/home/example):

```
/home/example/
├── PAT0/
│   ├── SERIES0
│   │   └── series0_file.nii.gz
│   ├── SERIES1
│   │   └── series1_file.nii.gz
│   └── ROI
│       ├── roi0_file.nii.gz
│       └── roi1_file.nii.gz
└── PAT1/
```

3.2 Script auto-completion with patient tree (v>=25.10.1)



Original script as follows:

```
LIFEx.Input.Directory=/home/example/
LIFEx.Output.Directory=/home/example/results
# *****
LIFEx.Series0.RelativeDirectory=SERIES0
LIFEx.Series0.Selected=true

LIFEx.Series0.Operation0=Series0Operation0Content
LIFEx.Series0.Operation1=Series0Operation1Content
LIFEx.Series0.Operation2=Series0Operation2Content

LIFEx.Roi0.Operation0=Roi0Operation0Content
LIFEx.Roi0.Operation1=Roi0Operation1Content
LIFEx.Roi0.Operation2=Roi0Operation2Content
# *****
LIFEx.Series1.RelativeDirectory=SERIES1
LIFEx.Series1.Selected=false

LIFEx.Series1.Operation0=Series1Operation0Content
LIFEx.Series1.Operation1=Series1Operation1Content
LIFEx.Series1.Operation2=Series1Operation2Content

LIFEx.Roi1.Operation0=Roi1Operation0Content
LIFEx.Roi1.Operation1=Roi1Operation1Content
LIFEx.Roi1.Operation2=Roi1Operation2Content
```

Auto-completion will write a new content script as follows:

```
LIFEx.Input.Directory=/home/example/
LIFEx.Output.Directory=/home/example/results
# *****
LIFEx.Patient0.Series0.Operation0=Series0Operation0Content
LIFEx.Patient0.Series0.Operation1=Series0Operation1Content
LIFEx.Patient0.Series0.Operation2=Series0Operation2Content
LIFEx.Patient0.Roi0.Operation0=Roi0Operation0Content
LIFEx.Patient0.Roi0.Operation1=Roi0Operation1Content
LIFEx.Patient0.Roi0.Operation2=Roi0Operation2Content
LIFEx.Patient0.Roi1.Operation0=Roi1Operation0Content
```

3.2 Script auto-completion with patient tree (v>=25.10.1)

```
LIFEx.Patient0.Roi1.Operation1=Roi1Operation1Content
LIFEx.Patient0.Roi1.Operation2=Roi1Operation2Content
LIFEx.Patient0.Roi0=PAT0/ROI/roi0_file.nii.gz
LIFEx.Patient0.Roi1=PAT0/ROI/roi1_file.nii.gz
LIFEx.Patient0.Series0=PAT0/SERIES0/series0_file.nii.gz
# *****
LIFEx.Patient1.Series0.Operation0=Series1Operation0Content
LIFEx.Patient1.Series0.Operation1=Series1Operation1Content
LIFEx.Patient1.Series0.Operation2=Series1Operation2Content
LIFEx.Patient1.Roi0.Operation0=Roi0Operation0Content
LIFEx.Patient1.Roi0.Operation1=Roi0Operation1Content
LIFEx.Patient1.Roi0.Operation2=Roi0Operation2Content
LIFEx.Patient1.Roi1.Operation0=Roi1Operation0Content
LIFEx.Patient1.Roi1.Operation1=Roi1Operation1Content
LIFEx.Patient1.Roi1.Operation2=Roi1Operation2Content
LIFEx.Patient1.Roi0=PAT1/ROI/roi0_file.nii.gz
LIFEx.Patient1.Roi1=PAT1/ROI/roi1_file.nii.gz
LIFEx.Patient1.Series0=PAT1/SERIES0/
```

• Example 3

Two patients, each with two series and two ROIs. Note that in this case, only one of the two series will be loaded - this introduces the concept of a selected series. The selection is inverted compared to Example 2.

Series operations

In this last example (the most complicated) we make you understand that the indices of the series declared in the original script can be modified in the automatically reconstructed script. Indeed, it is impossible for the application to have a series index "Series1" when "Series0" does not exist. This would be the case here, because Series0 is selected=false. In this case, the automatic reconstruction allows itself to reindex the indices so that it always starts at 0. This indicates that the correspondence with the original script must be done with caution and knowing this.

File tree example (Tree structure identical to that of Example 2) (/home/example):

```
/home/example/
├── PAT0/
│   ├── SERIES0
│   │   └── series0_file.nii.gz
│   ├── SERIES1
│   │   └── series1_file.nii.gz
│   └── ROI
│       ├── roi0_file.nii.gz
│       └── roi1_file.nii.gz
├── PAT1/
│   ├── SERIES0
│   │   ├── series0_file0.dcm
│   │   ├── series0_file1.dcm
│   │   ├── series0_file2.dcm
│   │   └── ...
│   ├── SERIES1
│   │   ├── series1_file0.dcm
│   │   ├── series1_file1.dcm
│   │   └── series1_file2.dcm
│   └── ROI
```

3.2 Script auto-completion with patient tree (v>=25.10.1)

```
|_ roi0_file.nii.gz
|_ roi1_file.nii.gz
```

Original script as follows:

```
LIFEx.Input.Directory=/home/example/
LIFEx.Output.Directory=/home/example/results
# *****
LIFEx.Series0.RelativeDirectory=SERIES0
LIFEx.Series0.Selected=false

LIFEx.Series0.Operation0=Series0Operation0Content
LIFEx.Series0.Operation1=Series0Operation1Content
LIFEx.Series0.Operation2=Series0Operation2Content

LIFEx.Roi0.Operation0=Roi0Operation0Content
LIFEx.Roi0.Operation1=Roi0Operation1Content
LIFEx.Roi0.Operation2=Roi0Operation2Content
# *****
LIFEx.Series1.RelativeDirectory=SERIES1
LIFEx.Series1.Selected=true

LIFEx.Series1.Operation0=Series1Operation0Content
LIFEx.Series1.Operation1=Series1Operation1Content
LIFEx.Series1.Operation2=Series1Operation2Content

LIFEx.Roi1.Operation0=Roi1Operation0Content
LIFEx.Roi1.Operation1=Roi1Operation1Content
LIFEx.Roi1.Operation2=Roi1Operation2Content
```

Auto-completion will write a new content script as follows:

```
LIFEx.Input.Directory=/home/example/
LIFEx.Output.Directory=/home/example/results
# *****
LIFEx.Patient0.Series0.Operation0=Series1Operation0Content
LIFEx.Patient0.Series0.Operation1=Series1Operation1Content
LIFEx.Patient0.Series0.Operation2=Series1Operation2Content
LIFEx.Patient0.Roi0.Operation0=Roi0Operation0Content
LIFEx.Patient0.Roi0.Operation1=Roi0Operation1Content
LIFEx.Patient0.Roi0.Operation2=Roi0Operation2Content
LIFEx.Patient0.Roi1.Operation0=Roi1Operation0Content
LIFEx.Patient0.Roi1.Operation1=Roi1Operation1Content
LIFEx.Patient0.Roi1.Operation2=Roi1Operation2Content
LIFEx.Patient0.Roi0=PAT0/ROI/roi0_file.nii.gz
LIFEx.Patient0.Roi1=PAT0/ROI/roi1_file.nii.gz
LIFEx.Patient0.Series0=PAT0/SERIES1/series1_file.nii.gz
# *****
LIFEx.Patient1.Series0.Operation0=Series1Operation0Content
LIFEx.Patient1.Series0.Operation1=Series1Operation1Content
LIFEx.Patient1.Series0.Operation2=Series1Operation2Content
LIFEx.Patient1.Roi0.Operation0=Roi0Operation0Content
LIFEx.Patient1.Roi0.Operation1=Roi0Operation1Content
```

```
LIFEx.Patient1.Roi0.Operation2=Roi0Operation2Content
LIFEx.Patient1.Roi1.Operation0=Roi1Operation0Content
LIFEx.Patient1.Roi1.Operation1=Roi1Operation1Content
LIFEx.Patient1.Roi1.Operation2=Roi1Operation2Content
LIFEx.Patient1.Roi0=PAT1/ROI/roi0_file.nii.gz
LIFEx.Patient1.Roi1=PAT1/ROI/roi1_file.nii.gz
LIFEx.Patient1.Series0=PAT1/SERIES1/
```

3.3 Series common operations

- Define a common output directory for the saving of all series:
LIFEx.Output.Directory={directory/subDirectory/}
- Define a global operation list to be applied across all loaded series:
LIFEx.Operation0=list of operations to be performed on all series
- Optionally open the results directory at script completion (default = false):
LIFEx.Output.Directory.OpenAtTheEnd=true

3.4 Series operations

Series operations

- Export anonymized series in DICOM format, provided original input is DICOM:
LIFEx.Patient0.Series0.Operation0=**Save anonymous dcm**
- Export series in DICOM format, provided original input is DICOM:
LIFEx.Patient0.Series0.Operation0=**Save dcm**
- Export series in NRRD format:
LIFEx.Patient0.Series0.Operation0=**Save nrrd**
- Export series in NiftI format (float 32 bits):
LIFEx.Patient0.Series0.Operation0=**Save nii float32**
- Export anonymized series in NiftI format (float 32 bits):
LIFEx.Patient0.Series0.Operation0=**Save Anonymous nii float32**
- Export series in NiftI format (uint 16 bits) (2 possibilities):
LIFEx.Patient0.Series0.Operation0=**Save nii uint16**
LIFEx.Patient0.Series0.Operation0=**Save nii**
- Export anonymized series in NiftI format (uint 16 bits):
LIFEx.Patient0.Series0.Operation0=**Save Anonymous nii uint16**
- Export series in ECAT format: (.v):
LIFEx.Patient0.Series0.Operation0=**Save ecats**
- Export MIP 3D Plans: Coronal, Sagittal, Axial in nifti format (float32):
LIFEx.Patient0.Series0.Operation0=**Save MIP 3D Plans**
- Export dynamic representation as video: in video format (mp4):
LIFEx.Patient0.Series0.Operation0=**Save mp4**
- Change output directory of Series saving:
LIFEx.Patient0.Series0.Operation0.Output.Directory={directory/subDirectory/}
- Unit of Y axis of series, choose between: kBq/mL || SUVbw || SUVlbm_morgan
|| SUVlbm_james || SUVlbm_janma || SUVibw || SUVbsa || Cpx/vx || Gy/vx || %/vx
|| .# class || .# k Pa || HU || T || mL/100g || mL/100g/min || sec || RC (.###) || min-1
|| Proba || # || ## || ### || #### || #####
LIFEx.Patient0.Series0.Operation0=**SUVbw**

3.4 Series operations

- Crop3D series. Coordinates are expressed in voxels (on referenced series). You can obtain these two crop coordinates in the interface via the Series/Crop menu and the following fields: "start corner" and "end corner".
LIFEx.Patient0.Series0.Operation0=**Crop3D,coorX1,coorY1,coorZ1,coorX2,coorY2,coorZ2**
- Crop2D series. Coordinates are expressed in voxels (on referenced series). You can obtain these two crop coordinates in the interface via the Series/Crop menu and the following fields: "start corner" and "end corner".
LIFEx.Patient0.Series0.Operation0=**Crop2D,coorX1,coorY1,coorX2,coorY2**
- Apply spatial resampling with specified voxel dimensions "SpatialResampling":
LIFEx.Patient0.Series0.Operation0=**SpatialResampling,xSpacing,ySpacing,zSpacing**
Example with spacing (x=3 mm, y=3mm, z=3mm):
LIFEx.Patient0.Series0.Operation0=SpatialResampling,3.0,3.0,3.0
- Perform linear combination of two series with user-defined coefficients:
LIFEx.Patient0.Series0.Operation0=**SeriesOperator,CoefA,SeriesA,OperatorName,CoefB,SeriesB**
OperatorName between: ADDITION || SUBTRACT || AVERAGE || DIVIDE || MULTIPLY
|| MAXIMUM || MINIMUM || AFFINE_TRANSFORM
Example of $S3 = 0.6 * S1 + 0.4 * S2$ then save S3 series result:
LIFEx.Patient0.Series0=S0file.nii.gz
LIFEx.Patient0.Series1=S1file.nii.gz
LIFEx.Patient0.Series0.Operation0=**SeriesOperator,0.6,S1,ADDITION,0.4,S2|SaveNii**
- Apply LaplacienOfGaussian spatial filtering with defined kernel size and dimensionality:
LIFEx.Patient0.Series0.Operation0=**LaplacienOfGaussianFilter,SigmaX,SigmaY,SigmaZ,CalculationDimension,WholeBody,PaddingMethod**
Example with Sigma (x=2 mm, y=2mm, z=2mm), 3d calculation dimension, wholebody enable and reflect padding method:
LIFEx.Patient0.Series0.Operation0=LaplacienOfGaussianFilter,2,2,2,3d,true,reflect
- float value Sigma is the FWHM/2 of kernels size in millimeter.
- 3d calculation dimension is between [2d, 3d]
- wholebody is always true in script
- padding method is between [reflect, periodic, edge, zero]
- Apply Gaussian spatial filtering with defined kernel size and dimensionality:
LIFEx.Patient0.Series0.Operation0=**GaussianFilter,SigmaX,SigmaY,SigmaZ,CalculationDimension,WholeBody,PaddingMethod**
Example with Sigma (x=2 mm, y=2mm, z=2mm), 3d calculation dimension, wholebody enable and reflect padding method:
LIFEx.Patient0.Series0.Operation0=GaussianFilter,2,2,2,3d,true,reflect
- float value Sigma is the FWHM/2 of kernels size in millimeter.
- 3d calculation dimension is between [2d, 3d]
- wholebody is always true in script
- padding method is between [reflect, periodic, edge, zero]
- Apply spatial mean filtering with defined kernel size and dimensionality:
LIFEx.Patient0.Series0.Operation0=**MeanFilter,kernelDiameterSizeVx,CalculationDimension,WholeBody,PaddingMethod**
Example with kernelDiameterSizeVx 3 vx, 3d calculation dimension, wholebody enable and reflect padding method:
LIFEx.Patient0.Series0.Operation0=MeanFilter,3,3d,true,reflect
- float value sigma is the FWHM/2 of kernels size in millimeter.
- integer value diameter kernel size between [3, 5, 7, 9, 11, 13, 15]
- 3d calculation dimension is between [2d, 3d]
- wholebody is always true in script

- padding method is between [reflect, periodic, edge, zero]
- Apply wavelet-based decomposition filter with user-defined parameters:
 LIFEx.Patient0.Series0.Operation0=**WaveletsFilter,TransformAlongZAxis,TransformAlongYXAxis,FamilyName,Order,Level,PaddingMethod**
 Example with TransformAlongZAxis enable, TransformAlongYXAxis enable, coiflet familyName, 1 order and 1 level
 LIFEx.Patient0.Series0.Operation0=WaveletsFilter,true,true,coiflets,1,1,reflect
 - transformAlongZAxis is between [true, false]
 - transformAlongYXAxis is always true
 - familyName is between [coiflets, biorthogonal, daubechies, haar, reverse biorthogonal, symlets]
 - daubechies order is between [1, ..., 38]
 - symlets order is between [2, ..., 20]
 - coiflets order is between [1, ..., 17]
 - reverse/biorthogonal order is between [11, 13, 15, 22, 24, 26, 28, 31, 33, 35, 37, 39, 44, 55, 68]
 - haar order is equal [1]
 - Level is equal [1]
 - padding method is between [reflect, periodic, edge, zero]
- Apply Laws' texture energy filters with selected kernel family:
 LIFEx.Patient0.Series0.Operation0=**LawsFilter,KernelSize,WholeBody,PaddingMethod**
 Example with (l3, l3, l3 Kernel size, WholeBody enable, and reflect Padding method
 LIFEx.Patient0.Series0.Operation0=LawsFilter,l3,l3,l3,true,reflect
 - kernel size is between [l3, l5, e3, e5, s3, s5, w5, r5]
 - with l3: level (1, 2, 1)
 - with l5: level (1, 4, 6, 4, 1)
 - with e3: edge (-1, 0, 1)
 - with e5: edge (-1, -2, 0, 2, 1)
 - with s3: spot (-1, 2, -1)
 - with s5: spot (-1, 0, 2, 0, -1)
 - with w5: wave (-1, 2, 0, -2, 1)
 - with r5: ripple (1, -4, 6, -4, 1)
 - wholebody is always true in script
 - padding method is between [reflect, periodic, edge, zero]

Chapter 4

ROI operations

4.1 Syntax examples

- Example: apply an absolute threshold between 2.5 and 50:
LIFEx.Patient0.Roi0.Operation0=n,2.5,50
- Example: absolute thresholding (2.5-50) followed by ROI export in NIfTI format:
LIFEx.Patient0.Roi0.Operation0=n,2.5,50|Save nii
- Example: subtraction of two loaded ROIs, followed by saving of the resulting ROI:
LIFEx.Patient0.Roi0.Operation0=SelectedAllRoi|Subtract|SelectedAllRoi|Save nii

4.2 ROI operations from threshold menu

- Apply absolute thresholding (min-max range):
LIFEx.Patient0.Roi0.Operation0=n,**ValueOfMinThreshold,ValueOfMaxThreshold**
- Apply relative thresholding based on percentage:
LIFEx.Patient0.Roi0.Operation0=n%,**ValueOfPercentThreshold**

4.3 ROI operations from file/edit menu

- Fixed 40% relative threshold:
LIFEx.Patient0.Roi0.Operation0=**40%**
- Fixed 70% relative threshold:
LIFEx.Patient0.Roi0.Operation0=**70%**
- Apply peak-based thresholding:
LIFEx.Patient0.Roi0.Operation0=**Peak**
- Apply Nestle method with specified threshold:
LIFEx.Patient0.Roi0.Operation0=**Nestle,ValueOfThreshold**
- Apply PERCIST thresholding (requires preloaded Liver ROI):
LIFEx.Patient0.Roi0.Operation0=**PERCIST**

4.3 ROI operations from file/edit menu

- Retain a single ROI from the loaded set
LIFEx.Patient0.Roi0.Operation0=**KeepOne**
- Divide ROI into separate components
LIFEx.Patient0.Roi0.Operation0=**Split**
- Union -> Union of all ROIs
LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi|Union**
- Intersection -> Intersection of ROIs
LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi|Intersection**
- Subtract -> Subtraction (with only 2 ROIs loaded)
LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi|Subtract**
- Morphological dilation with kernel size n voxels
LIFEx.Patient0.Roi0.Operation0=**Dilate,n**
- n is the number of voxels of length of kernel (vx)
- Morphological erosion with kernel size n voxels
LIFEx.Patient0.Roi0.Operation0=**Erode,n**
- n is the number of voxels of length of kernel (vx)
- Ring ROI operation generates a hollow structure of thickness n mm (half applied as lateral thickness). Example: *Ring, 10* produces a 10 mm kernel with 5 mm lateral extent.
LIFEx.Patient0.Roi0.Operation0=**Ring,n**
Example with 10 millimeters (= 5 millimeters of lateral extent): LIFEx.Patient0.Roi0.Operation0=**Ring,10**
- Select all loaded ROIs
LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi**
- Export ROI as Nift1 file:
LIFEx.Patient0.Roi0.Operation0=**Save nii**
- Export ROI as DICOM file:
LIFEx.Patient0.Roi0.Operation0=**Save dcm**
- Export ROI as CSV (comma-separated values):
LIFEx.Patient0.Roi0.Operation0=**Save csv**
- Export ROI in Turku PET Centre format (.dft):
LIFEx.Patient0.Roi0.Operation0=**Save dft**
- Placeholder for Recovery Coefficient export (not implemented):
LIFEx.Patient0.Roi0.Operation0=**Save dftrc**

4.4 ROI operations from measure menu

- Export ROI as NRRD (Pmod-compatible):
LIFEx.Patient0.Roi0.Operation0=**Save nrrd**
- Define output directory for ROI saving:
LIFEx.Patient0.Roi0.Operation0.**Output.Directory**={directory/subDirectory/}
- Define a global output directory for saving all ROIs:
LIFEx.**Output.Directory**={directory/subDirectory/}

4.4 ROI operations from measure menu

- Compare ROI operation generates a comparison result file. The comparison is carried out two by two, on all the ROIs loaded in the application.
LIFEx.Patient0.Roi0.Operation0=**Compare**

4.5 ROI operations from sort menu

- Quarter division for labeling ROI: LIFEx.Patient0.Roi0.Operation0="**Quarter Division**"
Complex example: LIFEx.Patient0.Roi0.Operation0=**Quarter Division|Selected All Roi|Save nii**

Chapter 5

Protocol operations

5.1 Protocol TEXTURE operations

5.1.1 Rationale

For large cohorts or multiple ROIs per patient, scripting enables automated extraction of indices (SUV, volume, histogram-based, textural) without user intervention. Prior ROI preparation is required. Scripts may be executed repeatedly with varied parameter sets to assess their influence on extracted features.

Example: For 100 patients each with 10 ROIs, the *nbGreyLevels* parameter may be varied (64, 128, 256) to evaluate its impact, requiring three separate script executions., see the "Can I run several script files at once?" p.11.

Another example: You have 10 ROI per patient and a set of 100 patients to be processed. You are interested in studying the impact of the *SpatialResampling* parameter on the 10*100 ROI, by setting this parameter to 2x2x2 mm and to 4x4x4 mm. You can write and run a first script by setting the *SpatialResampling* parameter to 2x2x2 mm. Then duplicate the script and change 2x2x2 mm by 4x4x4 mm to produce a second script to be run. Running the two scripts will calculate all indices for you automatically and store them in csv files. Setting the spatial resampling parameter to 2x2x2 mm in the script file can be done as in series operations chapter.

5.1 Protocol TEXTURE operations

A preliminary ROI verification (CheckTex) ensures suitability for texture analysis, avoiding time-consuming invalid computations. This will check whether the ROI includes a single cluster and contain a number of voxels greater than that required for consistent textural feature calculation. If one of these two conditions is not met, a warning message will be displayed.

All these lines contain the series paths, ROIs and parameters needed to redo the texture analysis you've just designed in the user interface. You can then drag this file into the application's series/images/scripts reading area to run the whole procedure. You can also use this file as a starting point for modifying certain parameters in order to re-execute a new analysis under the same conditions.

5.1.2 Explanation of field contents

Check: Setting Check=true enables a fast preliminary computation restricted to simple features (voxel count, min, mean, max).

KeepTheLargestCluster: The KeepTheLargestCluster option retains only the largest connected component within an ROI, discarding smaller clusters.

IsEnabledRoiUnion: merges all segmented regions into a single ROI prior to texture calculation.

BinSize and NbGreyLevels (how set the quantization): Grey-level discretization can be parameterized via BinSize or NbGreyLevels, with automatic computation of the unset parameter.

For instance:

if $BinSize = 0$ then $BinSize = (boundMax - boundMin) / (nbGreyLevels - 1)$
if $nbGreyLevels = 0$ then $nbGreyLevels = ((boundMax - boundMin) / binSize) + 1$
if $BinSize = 0$ and $nbGreyLevels = 0$ then $nbGreyLevels = 64$ by default
 $boundMin$ and $boundMax$ are the minimum and maximum values in the processed ROI.

Discretization has been set with:

$$discretizedValue = \text{floor}\left(\left(nbGreyLevels * \frac{originalValue - boundMin}{boundMax - boundMin}\right)\right) + 1$$

• Have the minimum and maximum bounds of a whole cohort of patients in a script (without texture calculation)

It is easy to find the BoundMax of all the ROI of all your patients in order to put this bound in the final script. This has to be done in 2 steps with exactly the same script (by changing one parameter). Here is the process.

- step 1: change this value to "true" on the Check key. It allows calculating the min, max bounds of all the ROI in a single pass of the script.
- step 2: run the script a first time to have only bound results; You then retrieve the max of the max of all the ROIs and correct the value of the MaxBound key in the script file
- step 3: change to false value of Check key in the script file
- step 4: run the script a second time to have all feature results

5.1.3 Automatic texture script creation

Since version 7.6.0, LIFEx is capable of automatically generating texture scripts based on GUI-defined protocols. To do so, follow the standard texture protocol in the interface. However, instead of executing the texture calculations by pressing the run button, you can generate the corresponding script by pressing the script button (to the left of the run button).

A text file will open with the contents of the script written for you, taking into account the interface's texture parameters. This script also includes the loading of previously opened Series and ROI.

The generated script can be reloaded into a new LIFEx session for automatic execution of texture analysis.

5.1.4 Operations

Operations between { } are optional. If they are added, do not place the braces { } in the script.

- Texture protocol execution using global ROI and absolute discretization scheme:

template: LIFEx.Patient0.Series0.Operation0=**Texture, KeepTheLargestCluster, Check, isEnabledRoiUnion, DistanceWithNeighbours, DimensionProcessing, Absolute, BinSize, NbGreyLevels, MinBound, MaxBound** {, **xSpatialResampling, ySpatialResampling, zSpatialResampling**} {, **filterName, ..., ...**} (see: filters in "Available Series operations" section)

example1: LIFEx.Patient0.Series0.Operation0=**Texture, true, false, false, 1, 3D, Absolute, 0, 64, 0, 20**
Texture with KeepTheLargestCluster=true
and with check=false
and with isEnabledRoiUnion=false
and with DistanceWithNeighbours=1 and with DimensionProcessing=3D
and with Absolute discretization
and with BinSize=0
and with NbGreyLevels=64
and with MinBound=0
and with MaxBound=20

example2: LIFEx.Patient0.Series0.Operation0=**Texture, true, false, false, 1, 3D, Absolute, 0, 64, 0, 20, 2.5, 2.5, 2.5**
Texture with KeepTheLargestCluster=true
and with check=false
and with isEnabledRoiUnion=false
and with DistanceWithNeighbours=1 and with DimensionProcessing=3D
and with Absolute discretization
and with BinSize=0
and with NbGreyLevels=64
and with MinBound=0
and with MaxBound=20
and with xSpatialResampling=2.5 mm
and with ySpatialResampling=2.5 mm
and with zSpatialResampling=2.5 mm

example3: LIFEx.Patient0.Series0.Operation0=**Texture, true, false, false, 1, 3D, Absolute, 0, 64, 0, 20, 2.5, 2.5, 2.5, LaplacianOfGaussianFilter, 2.000, 2.000, 2.000, 3d, WholeBody, reflect**
Texture with KeepTheLargestCluster=true
and with check=false
and with isEnabledRoiUnion=false
and with DistanceWithNeighbours=1 and with DimensionProcessing=3D
and with Absolute discretization
and with BinSize=0
and with NbGreyLevels=64
and with MinBound=0
and with MaxBound=20
and with xSpatialResampling=2.5 mm

5.1 Protocol TEXTURE operations

and with ySpatialResampling=2.5 mm
and with zSpatialResampling=2.5 mm
and with LaplacianOfGaussianFilter filter

- Execution with relative mean-standard deviation discretization strategy (global ROI):

template: LIFEx.Patient0.Series0.Operation0=**Texture, KeepTheLargestCluster, Check, isEnabledRoiUnion DistanceWithNeighbours, DimensionProcessing, RelativeMeanSd, BinSize, NbGreyLevels, MinBound, MaxBound** {,xSpatialResampling,ySpatialResampling,zSpatialResampling} {,filterName, ..., ...} (see: filters in "Available Series operations" section)

example: LIFEx.Patient0.Series0.Operation0=**Texture,true,false,false,1,3D,RelativeMeanSd,0,64**
Texture with KeepTheLargestCluster=true, check=false, isEnabledRoiUnion=false
and with DistanceWithNeighbours=1
and with DimensionProcessing=3D
and with RelativeMeanSd discretization
and with BinSize=0
and with NbGreyLevels=64

- Execution with relative min-max discretization strategy (global ROI):

template: LIFEx.Patient0.Series0.Operation0=**Texture, KeepTheLargestCluster, Check, isEnabledRoiUnion DistanceWithNeighbours, DimensionProcessing, RelativeMinMax, BinSize, NbGreyLevels, MinBound, MaxBound** {,xSpatialResampling,ySpatialResampling,zSpatialResampling} {,filterName, ..., ...} (see: filters in "Available Series operations" section)

example: LIFEx.Patient0.Series0.Operation0=**Texture,true,false,false,1,3D,RelativeMinMax,0,64**
Texture with KeepTheLargestCluster=true, check=false, isEnabledRoiUnion=false
and with DistanceWithNeighbours=1
and with DimensionProcessing=3D
and with RelativeMinMax discretization
and with BinSize=0
and with NbGreyLevels=64

- TextureMap protocol enables voxel-wise feature extraction with kernel-based processing and specified model (WholeBody, VoxelsOfRoi, BoundingBoxOfRoi) (local Map):

template: LIFEx.Patient0.Series0.Operation0=**TextureMap, KernelProcessing, MapModel, BinSize, NbGreyLevels, MinBound, MaxBound**
with KernelProcessing between 3||5||7
with MapModel between WholeBody||VoxelsOfRoi||BoundingBoxOfRoi
Don't forget the ROI loading line if you're using MapModel VoxelsOfRoi||BoundingBoxOfRoi

example: LIFEx.Patient0.Series0.Operation0=**TextureMap,3,WholeBody,0,64,0,20**
TextureMap with KernelProcessing=3
and with MapModel=WholeBody
and with BinSize=0

and with NbGreyLevels=64
and with MinBound=0
and with MaxBound=20

5.2 MTV Protocol operations

5.2.1 Introduction

MTV (Metabolic Tumor Volume) computation can be automated through scripts, chaining image and ROI loading into consolidated .csv outputs.

5.2.2 Operations

- MTV protocol execution is invoked via the operation Mtv, KeepTheLargestCluster.: template:
LIFEx.Patient0.Series0.Operation0=**Mtv,KeepTheLargestCluster**
example: LIFEx.Patient0.Series0.Operation0=**Mtv,false**
- KeepTheLargestCluster on MTV: defines whether only the largest cluster should be kept when an ROI contains several clusters.
If the value of this key is true, then only the largest cluster is kept (the others are deleted during the calculation).
If this value is false, then the ROI remains composed of several clusters if this is the case.
-> In the MTV protocol, we strongly recommend leaving this parameter at **false**, otherwise all clusters will not be taken into account when calculating the MTV (this depends mainly on the goal set).

5.2.3 Script example

This section shows a script example of MTV protocol.

You can copy/paste this script into a text file named script.txt (for example). Don't forget to change some information: ex. directory/subDirectory

Protocol operations

5.2 MTV Protocol operations

```
# result file -> mandatory
LIFEx.Output.Directory={directory/subDirectory}

#####
# Patient0 / Series / ROI
#####
# loading series -> mandatory
LIFEx.Patient0.Series0={directory/subDirectory}/PT0
LIFEx.Patient0.Series0.Operation0=Mtv,false
# loading ROI -> mandatory
LIFEx.Patient0.Roi0={directory/subDirectory}/RoiVolume/R1.uint16.nii.gz
LIFEx.Patient0.Roi1={directory/subDirectory}/RoiVolume/R2.uint16.nii.gz

#####
# Patient 1 / Series
#####
# loading series -> mandatory
LIFEx.Patient1.Series0={directory/subDirectory}/PT1
LIFEx.Patient1.Series0.Operation0=Mtv,false
# loading ROI -> mandatory
LIFEx.Patient1.Roi0={directory/subDirectory}/RoiVolume/R1.uint16.nii.gz
LIFEx.Patient1.Roi1={directory/subDirectory}/RoiVolume/R2.uint16.nii.gz
```

5.2.4 Output files

The root value of the key "*LIFEx.Output.Directory*" will be used for the construction of 2 files (2 files MTV_ROI and 2 files MTV_SUMMARY) will be saved at the end of the script execution by the application:

- MTV_ROI_results.csv: will contain all the features extracted from each ROI;
- MTV_ROI_legend.csv: will contain the legend of all features extracted from each ROI
- MTV_SUMMARY_results.csv: will contain aggregated features calculated from several ROIs.
- MTV_SUMMARY_legend.csv: will contain the legend of aggregated features calculated from several ROIs.