Scripts

Local Image Features Extraction
— LIFEx —

C. Nioche, F. Orlhac, I. Buvat

LIFEx version 7.7.8,
Last update of document: 2025/02/24

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
common US Properties
19

Series operations
Syntax examples
Available Series common operations
Available Series operations
21

ROI operations
Syntax examples
Available ROI operations [threshold menu]
Available ROI operations [file/edit menu]
Available ROI operations [edit/reformat menu]
25

Protocol operations
Protocol TEXTURE operations
MTV Protocol operations
29

List of Figures

Part I
What is a script?

Chapter 1
Introduction

1.1 Rationale and what is a script?

Scripting is the process of writing scripts that automate certain tasks within LIFEx application. Scripting is often used in this application to automate repetitive tasks, perform complex calculations, or customize the user experience.

In other words, a script file makes it possible to run all operations and calculations without any user interaction. You can prepare it in advance. You can have as many script files as you want, with names that you choose and you can save them and modify them. They are simple text files.

The scripting procedure for LIFEx application typically involves the following steps:

- Writing the script: you can start writing the script itself. This involves using the syntax and commands of the scripting language to create a set of instructions that the application can execute.

- Debugging and testing the script: After writing the script, you will need to test it to make sure it works as intended. This involves running the script and checking its output, and debugging any errors or issues that may arise.

- Integrating/Executing the script with the application: Once the script is working correctly, you will need to execute it with the application. This involves specifying when and how the script should be executed within the application.

- Maintaining the script: Finally, you will need to maintain the script over time to ensure that it continues to work correctly as the application evolves and changes. This may involve updating the script to work with new versions of the application, fixing any bugs that arise, or adding new functionalities as needed.

1.2 General information about writing a script text file

Introduction: A script file for LIFEx is a text file containing a sequence of properties. Each property is written on a line.

Properties: Properties are configuration values managed as key=value pairs. In each pair, the key and value are both String values. The key identifies and is used to retrieve, the value, much as a variable name is used to retrieve the variable's value.

Property example: For example, an application capable of saving file might use a property named *"LIFEx.Output.Directory"* to keep track of the directory used for the saving the file.

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

- 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: a key is unique and cannot be repeated. Only one key per line.

- spaces before the key do not count.

- the order of the lines does not matter, however for human reading it is better to categorize the actions.

- file paths must always be written with a file separator "/" even under Windows.

- lines beginning with "#" are comments and are not interpreted by the application.

1.3 Script execution sequence

Here is the (mandatory) tasks order of execution in a script. Each script file will be read in full and executed as follows:

1. Series loading [mandatory]
2. — Series operations [optional]
3. — Series saving [optional]
4. ROI loading [optional]
5. — ROI operations [mandatory if "loaded ROI", otherwise optional]
6. — ROI saving [mandatory if "loaded ROI", otherwise optional]
7. Protocol loading [optional]
8. — MTV protocol operations [optional]
9. — Texture protocol operations [optional]
10. — ROI close [optional and automatic]
11. ROI close [optional and automatic]
12. Series close [automatic]

1.4 How to run a script file

- Drag the script file in the panel used to load patient images.

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

You can indeed run several script files one after another automatically. To do so, please select all script files and drag them in the panel used to put the patient image files. The script files will be executed one after another.

Part II
General concepts

Chapter 1
How to create a script?

1.1 Write a script

It is possible to chain the reading of the images and their respective ROI in order to obtain a file including all results (.csv).

1.2 Script for getting started

This section shows an script example. This script loads 2 series named PT0 and PT1, and then 2 ROI associated with PT0 (in Patient 0) and 2 ROI associated with PT1 (in Patient 1).

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

```
#_____________________________________________________________________
#
# 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

See below for a complete script example of save anonymous DICOM Series from 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 can be in NIfTI-1 format (.nii or .nii.gz). In this case, the complete pathway to the patient image file should be given in the script;
  - to load DICOM images, you must define the root directory of these images (without the final filename). All files included in this directory will be loaded;
- What are the ROI formats that can be managed using LIFEx scripts?
  - the NIfTI-1 and RTStruct are supported in LIFEx scripts. In both cases, the file name should include the full path to the file. In the case of RTStuct, all ROI are loaded without exception.
- Syntax of all pathways of files in LIFEx scripts:
  - the syntax of pathway must neither contain accents and nor space.
  - the syntax of pathway must not necessarily conform to your system rules:
    * Windows: Unit:/Directory/File.extension
      or Unit:/Your Directory/ for series of DICOM images
      example C:/Home/Users1/File1
    * Linux: /Your Directory/Your File.extension
    * MacOs: /Your Directory/Your File.extension

1.5 Perform operations

Some intermediate operations are possible on the ROI. They will be performed after reading the Series and ROI but before extracting the features and recording the results:

Main syntax of one operation: The syntax property for one operation is:

```
{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: It is possible to chain actions on the same line in order to retrieve the result of the previous action for the next action.

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

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

On this line the order (from left to right) of the operations is respected during execution.

Here's an example:

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

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.

How to create?

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=...

1.6 Perform common setting

LIFEx.Password: allows you to unlock a private function. Passwords are often used in our team to prepare new features that are not yet released. This allows the release of the application with hidden parts.
LIFEx.Password=****

LIFEx.GuiUpdate: showing (true) or not showing (false) of graphical user interface (is java headless) (LIFEx>=7.4.1) [true (default) || false]
This key allows to speed up the calculation of the scripts by not displaying the graphical user interface during the execution of the texture calculations (convenient for servers).
LIFEx.GuiUpdate=false||true

LIFEx.SeriesRoiGuiUpdate: showing (true) or not showing (false) of series/ROI images in graphical user interface (LIFEx>=7.5.6) [true (default) || false]
This key allows to speed up the calculation of the scripts by not displaying the different ROI and corresponding planar slices during the execution of the texture calculations.
LIFEx.SeriesRoiGuiUpdate=false||true

LIFEx.MessagesGuiUpdate: showing (true) or not showing (false) of messages in graphical user interface (LIFEx>=7.5.6) [true (default) || false]
This key allows to speed up the calculation of the scripts by not displaying the messages during the execution of the texture calculations.
LIFEx.MessagesGuiUpdate=false||true

Chapter 2
Common properties

2.1 common US Properties

add US Properties:

When reading US dicom, it is possible to read only the average signal by deactivating red/green/blue channel reading. These 3 channels are activated by default.

Add these lines to disable the 3 channels:
LIFEx.PropertiesSeriesUS.haveRedChannel=false
LIFEx.PropertiesSeriesUS.haveGreenChannel=false
LIFEx.PropertiesSeriesUS.haveBlueChannel=false

Chapter 3
Series operations

3.1 Syntax examples

- Saving series with anonymous DICOM field:
  LIFEx.Patient0.Series0.Operation0=Save anonymous dcm
- Change unit of series in SUVbw:
  LIFEx.Patient0.Series0.Operation0=SUVbw
- 2x2x2 Resampling series then saving result series in nifti format:
  LIFEx.Patient0.Series0.Operation0=ResampleRescaling,2,2,2|save nii uint16
- Saving result series in anonymous nifti format:
  LIFEx.Patient0.Series0.Operation0=Save Anonymous nii uint16
- Change output directory of series saving:
  LIFEx.Output.Directory={directory/subDirectory/}

3.2 Available Series common operations

- Change common output directory of all Series saving:
  LIFEx.Output.Directory={directory/subDirectory/}

- set the operation list on all series loaded:
  LIFEx.Operation0=list of operations to be performed on all series
- have an open result directory at the end (false is the default value or if the line does not exist):
  LIFEx.Output.Directory.OpenAtTheEnd=true

3.3 Available Series operations

- Save anonymous series in DICOM format (only if loaded series is in DICOM format too):
  LIFEx.Patient0.Series0.Operation0=**Save anonymous dcm**
- Save series in DICOM format (only if loaded series is in DICOM format too):
  LIFEx.Patient0.Series0.Operation0=**Save dcm**
- Save series in nrrd format:
  LIFEx.Patient0.Series0.Operation0=**Save nrrd**
- Save series in nifti format (float 32 bits):
  LIFEx.Patient0.Series0.Operation0=**Save nii float32**
- Save series in anonymous nifti format (float 32 bits):
  LIFEx.Patient0.Series0.Operation0=**Save Anonymous nii float32**

- Save series in nifti format (uint 16 bits) (2 posibilities):
  LIFEx.Patient0.Series0.Operation0=**Save nii uint16**
  LIFEx.Patient0.Series0.Operation0=**Save nii**
- Save series in anonymous nii (uint 16 bits):
  LIFEx.Patient0.Series0.Operation0=**Save Anonymous nii uint16**
- Save series in ECAT format: (.v):
  LIFEx.Patient0.Series0.Operation0=**Save ecat**
- Save MIP 3D Plans: Coronal, Sagittal, Axial in nifti format (float32):
  LIFEx.Patient0.Series0.Operation0=**Save MIP 3D Plans**
- Save mp4: 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 || 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**
- 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 a spatial resample rescaling "ResampleRescaling":
  LIFEx.Patient0.Series0.Operation0=**ResampleRescaling,SigmaX,SigmaY,SigmaZ**
  Example with Sigma (x=3 mm, y=3mm, z=3mm):
  LIFEx.Patient0.Series0.Operation0=ResampleRescaling,3.0,3.0,3.0
  - float value Sigma is the FWHM/2 of kernels size in millimeter.

- Apply a spatial "LaplacienOfGaussian" filter:
  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 a spatial "Gaussian" filter:
  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 a spatial "Mean" filter:
  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 a spatial "Wavelets" filter:
  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 a spatial "Laws" filter:
  LIFEx.Patient0.Series0.Operation0=**LawsFilter,KernelSize,WholeBody,PaddindMethod**

Example with (l3, l3, l3 Kernel size, WholeBody enable, and reflect Paddind 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

- Below is an example of an "Absolute threshold" (n) with MinThreshold=2.5 and MaxThreshold=50:
  LIFEx.Patient0.Roi0.Operation0=n,2.5,50

- Below is an other example of a "Absolute threshold" (n) with MinThreshold=2.5 and MaxThreshold=50 followed by the saving of result ROI in nifti (.nii) format:
  LIFEx.Patient0.Roi0.Operation0=n,2.5,50|Save nii

- Here is an example to subtract 2 loaded ROIs, then save the resulting subtractions:
  LIFEx.Patient0.Roi0.Operation0=SelectedAllRoi|Subtract|SelectedAllRoi|Save nii

4.2 Available ROI operations [threshold menu]

- Absolute threshold:
  LIFEx.Patient0.Roi0.Operation0=**n,ValueOfMinThreshold,ValueOfMaxThreshold**

- Percent threshold:
  LIFEx.Patient0.Roi0.Operation0=**n%,ValueOfPercentThreshold**
- 40 Percent threshold:
  LIFEx.Patient0.Roi0.Operation0=**40%**
- 70 Percent threshold:
  LIFEx.Patient0.Roi0.Operation0=**70%**
- Peak:
  LIFEx.Patient0.Roi0.Operation0=**Peak**
- Nestle threshold:
  LIFEx.Patient0.Roi0.Operation0=**Nestle,ValueOfThreshold**
- PERCIST threshold (need to have previously loaded a ROI named Liver):
  LIFEx.Patient0.Roi0.Operation0=**PERCIST**

4.3 Available ROI operations [file/edit menu]

- KeepOne
  LIFEx.Patient0.Roi0.Operation0=**KeepOne**

- Split
  LIFEx.Patient0.Roi0.Operation0=**Split**

- Union
  LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi|Union**

- Intersection
  LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi|Intersection**

- Subtract (with only 2 ROIs loaded)
  LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi|Subtract**

- Dilate
  LIFEx.Patient0.Roi0.Operation0=**Dilate,n**
  - n is the number of voxels of length of kernel (vx)
- Erode
  LIFEx.Patient0.Roi0.Operation0=**Erode,n**
  - n is the number of voxels of length of kernel (vx)
- Selected all ROI
  LIFEx.Patient0.Roi0.Operation0=**SelectedAllRoi**
- Save ROI in nifti (.nii) format file:
  LIFEx.Patient0.Rio0.Operation0=**Save nii**
- Save ROI in DICOM (.dcm) format file:
  LIFEx.Patient0.Roi0.Operation0=**Save dcm**
- Save ROI in comma separator value (.csv) format file:
  LIFEx.Patient0.Roi0.Operation0=**Save csv**
- Save ROI in turkupetcentre (.dft) format file:
  LIFEx.Patient0.Roi0.Operation0=**Save dft**
- Not yet implemented (request of Recovery Coefficient not available in script):
  LIFEx.Patient0.Roi0.Operation0=**Save dftrc**
- Save ROI in Pmod (.nrdd) format file:
  LIFEx.Patient0.Roi0.Operation0=**Save nrrd**

- Change output directory of ROI saving:
  LIFEx.Patient0.Roi0.Operation0.**Output.Directory**={directory/subDirectory/}
- Change common output directory of all ROI saving:
  LIFEx.**Output.Directory**={directory/subDirectory/}

4.4 Available ROI operations [edit/reformat menu]

- Ring ROI: with n millimeter length kernel ($n/2$ of lateral thickness)
  LIFEx.Patient0.Roi0.Operation0=**Ring,n**
  Example with 10 millimeters (= 5 millimeters of lateral thickness): LIFEx.Patient0.Roi0.Operation0=**Ring,10**

Chapter 5
Protocol operations

5.1 Protocol TEXTURE operations

5.1.1 Rationale

When you have a large number of ROI and/or a large number of patients for which you want to calculate usual indices (SUV, Volume, ...) and/or histogram-based or textural features, it can be convenient to run a script that will do all the calculations for you without any user interaction. ROI have to be prepared beforehand. You can then run the script many times if you want to perform the calculations using different sets of parameters.

For instance: You have 10 ROI per patient and a set of 100 patients to be processed. You are interested in studying the impact of the $nbGreyLevels$ parameter on the 10*100 ROI, by setting this parameter to 64, 128 and 256. You can create three script files that will all be identical except for the line that sets the $nbGreyLevels$ parameter. Running the script will calculate all indices for you automatically and store them in csv files. You will be able to run all three scripts using a single command line, 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 $ResampleRescaling$ 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

$ResampleRescaling$ 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.

Using scripts may produce numerous ROIs. To check that the resulting ROI are compatible with texture index calculation and before running such calculations that can be time-consuming, you can use the CheckTex button after the ROI creation and before proceeding with textural index calculation. 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: if true, allows to define a fast calculation of the script with only a few pre-selected features (voxelsCounting, min, mean, max). This key is used to produce a fast 1st pass on a patient cohort in order to establish the min and max values of all ROIs of all patients. These values can be entered in the same script for a 2nd pass (with texture calculation : Check=false).

KeepTheLargestCluster: allows you to define if you want to keep the largest of the clusters only when an ROI is composed of 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.

IsEnabledRoiUnion: is used to assemble all regions into a single (non connected) one before calculating texture features.

BinSize and NbGreyLevels (how set the quantization): When you set the quantization of grey-level discretization parameters, you can leave one of the two sets to 0. Its value will then be automatically calculated based on the other parameter setting.

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 $bounsMax$ are the minimum and maximum values in the processed ROI.

Discretization has been set with:

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

- **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

LIFEx (version>=7.6.0) may automatically create a simple texture script for you. 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.

Simply drag this script file into a new LIFEx session in the image loading section to launch your texture analysis automatically.

5.1.4 Operations

- running texture protocol (global ROI) with Absolute discretization:

  template: LIFEx.Patient0.Series0.Operation0=**Texture,**
  KeepTheLargestCluster,Check,isEnabledRoiUnion
  DistanceWithNeighbours,DimensionProcessing,
  Absolute,BinSize,NbGreyLevels, MinBound,MaxBound

  example: 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

- running texture protocol (global ROI) with RelativeMeanSd discretization:

  template: LIFEx.Patient0.Series0.Operation0=**Texture,**
  KeepTheLargestCluster,Check,isEnabledRoiUnion
  DistanceWithNeighbours,DimensionProcessing,
  RelativeMeanSd,BinSize,NbGreyLevels, MinBound,MaxBound

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

and with BinSize=0
and with NbGreyLevels=64

- running texture protocol (global ROI) with RelativeMinMax discretization:

  template: LIFEx.Patient0.Series0.Operation0=**Texture,**
  KeepTheLargestCluster,Check,isEnabledRoiUnion
  DistanceWithNeighbours,DimensionProcessing,
  RelativeMinMax,BinSize,NbGreyLevels, MinBound,MaxBound

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

- running texture protocol (local Map = Voxel-wise feature extraction) with Absolute discretization (mandatory):

  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

The calculation of MTV values can be automated with scripts. It is possible to chain the reading of the images and their respective ROI in order to obtain a file including all results (.csv).

5.2.2 Operations

- running MTV protocol: template: LIFEx.Patient0.Series0.Operation0=**Mtv,KeepTheLargestCluster**
  example: LIFEx.Patient0.Series0.Operation0=**Mtv,false**

- KeepTheLargestCluster on MTV: allows you to define if you want to keep the largest of the clusters only when an ROI is composed of 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

```
# 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.