

# How to make an own plugin with GUI for Microscopy Image Browser (version 2.x) (MIB version 2.x) Ilya Belevich

This tutorial demonstrates how to write a GUI (Graphical User Interface) based plugin and connect it to Microscopy Image Browser (MIB). Note! For MIB version 2.0 and later!

Task: write a GUI add-on that allows 1) crop, 2) resize, 3) invert, and 4) convert images to the *uint16* class.

Reference: the files described in this tutorial located in directory:

```
mib\Plugins\Tutorials\GuiTutorial\
```

## PART 1: Design of GUI

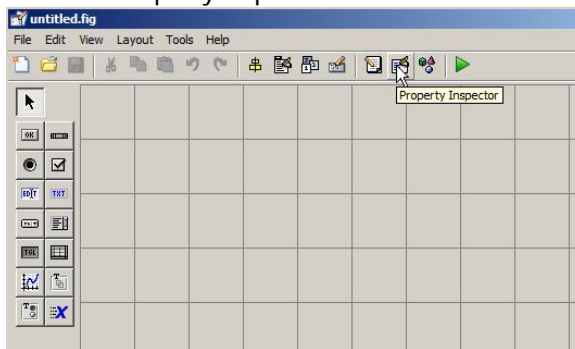
1. To create GUI we will use GUIDE program of Matlab. Type 'guide' in the Matlab command window to start it.

2. Select *Create New GUI->Blank GUI (Default)*.

Design the view of the add-on GUI (please refer for GUIDE details in Matlab help).

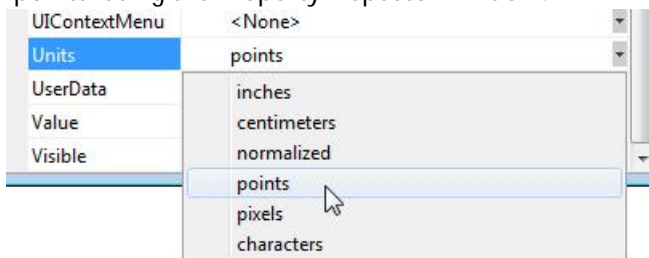
Few notes about design:

- Start the Property Inspector from the GUIDE toolbar or by a double mouse click on the grid.

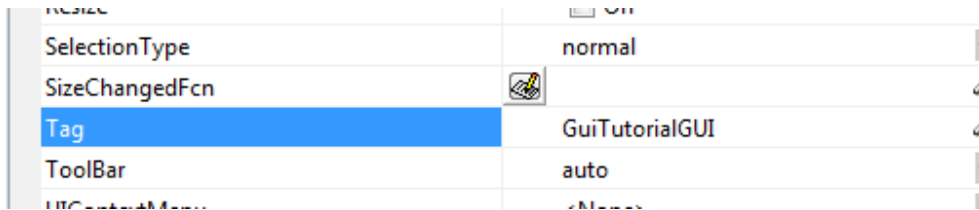


- Important!

For proper sizing of widgets across different platforms set 'Units' for all placed widgets to 'points' using the Property Inspector window!



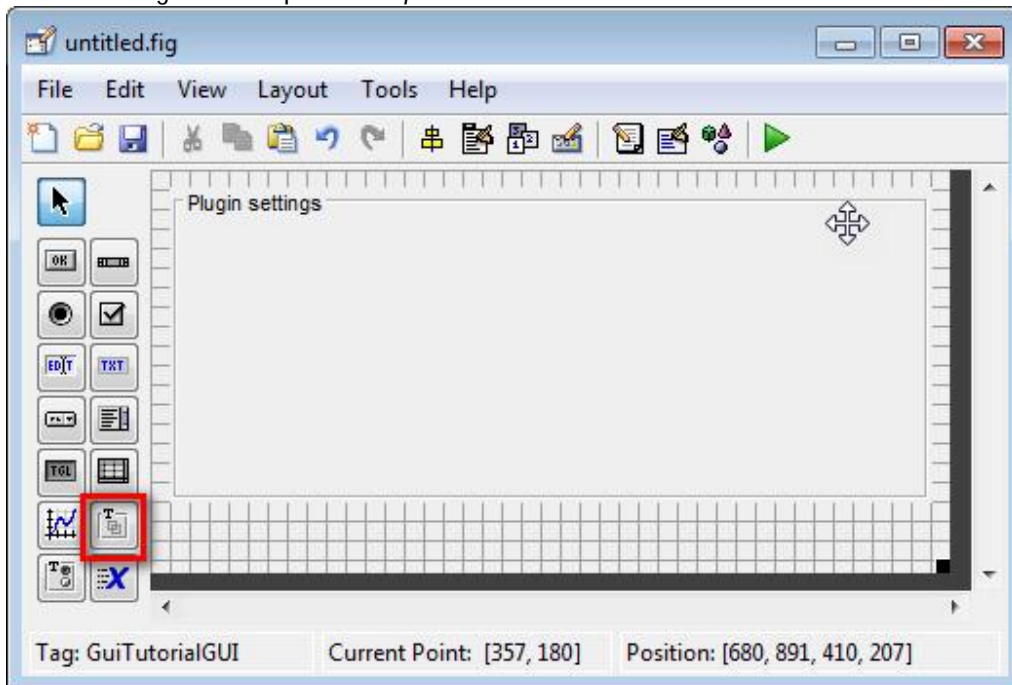
3. Assign a tag of for the plugin window: set the *Tag* of the main figure to *GuiTutorialGUI* in the Inspector window: *Inspector->Tag-> GuiTutorialGUI*



- add a title for the plugin: *Inspector->Name->Gui Example*
- change units to points: *Inspector->Units->Points*

4. Create a panel and add title: 'Plugin settings' (*Inspector -> Title -> Plugin settings*).

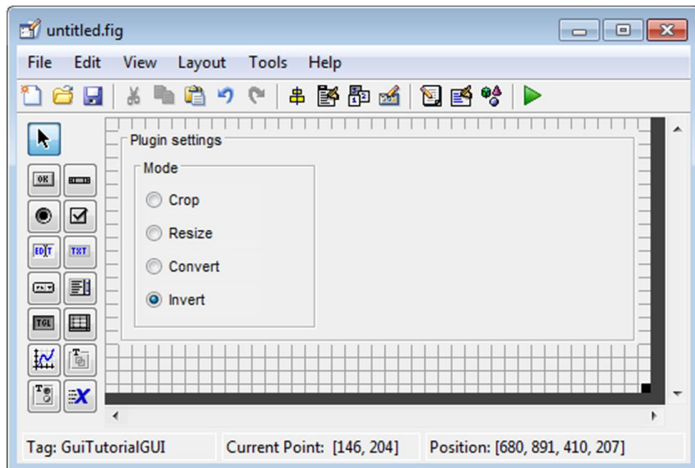
- change units to points: *Inspector->Units->Points*







5. Add a Radio Group  to the Plugin settings panel

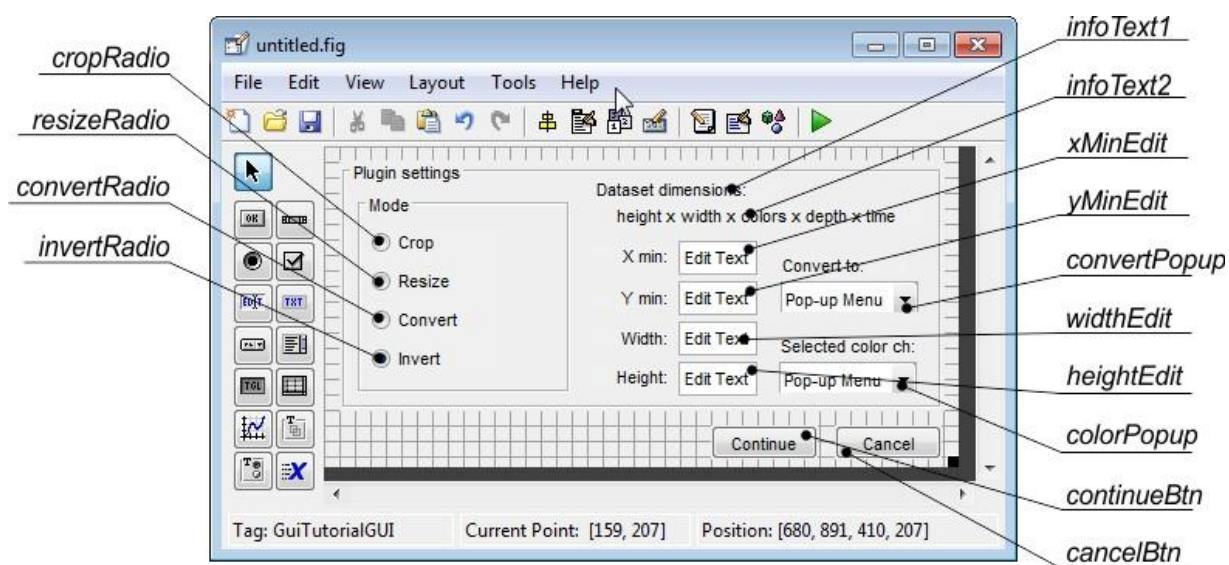
- assign its Title to Mode: *Inspector->Title->Mode*
- tag to buttonGroup: *Inspector-> Tag->buttonGroup*
- change units to points: *Inspector->Units->Points*

6. Add to the group 4 radio buttons (*Inspector->String: "Crop", "Resize", "Convert", "Invert"*) tagged (*Inspector->Tag*) as *cropRadio*, *resizeRadio*, *convertRadio*, and *invertRadio* respectively. Change units to points.



7. Add additional components:

<p>- 4 edit boxes:</p> 	<p><i>Inspector-&gt;Tag: xminEdit, yminEdit, widthEdit, heightEdit</i>  <i>Inspector-&gt;CreateFcn: remove %automatic text</i>  <i>Inspector-&gt;Units: points</i></p>
<p>- 2 popup menus</p> 	<p><i>Inspector-&gt;Tag: convertPopup, colorPopup</i>  <i>Inspector-&gt;CreateFcn: remove %automatic text</i>  <i>Inspector-&gt;Callback: remove %automatic text</i>  <i>Inspector-&gt;Units: points</i></p>
<p>- text boxes</p> 	<p><i>Inspector-&gt;Tag: see in the snapshot below</i>  <i>Inspector-&gt;String: see in the snapshot below</i>  <i>Inspector-&gt;Units: points</i></p>
<p>- 2 push buttons</p> 	<p><i>Inspector-&gt;Tag: continueBtn, cancelBtn</i>  <i>Inspector-&gt;String: Continue, Cancel</i>  <i>Inspector-&gt;Units: points</i></p>



8. Save figure and matlab code (GUIDE->Menu->Save as...) to the MIB\Plugins\Tutorials\GuiTutorialGUI directory

For example, the full path may look like this:

C:\Scripts\mib\Plugins\Tutorials\GuiTutorial\GuiTutorialGUI.fig

C:\Scripts\mib\Plugins\Tutorials\GuiTutorial\GuiTutorialGUI.m

9. Create CloseRequestFcn:

- double click with the left mouse button on any free space of the main window

- *Inspector* -> *CloseRequestFcn* -> Press  button to generate a template for the function in GuiTutorialGUI.m file

10. Create callback for the `buttonGroup` that is going to be used to define the function mode: Crop, Resize, Convert or Invert.

- double click on the Mode group with the left mouse button

- *Inspector* -> *SelectionChangedFcn* -> Press  button to generate a common callback for all radio buttons

12. Make sure that GuiTutorialGUI.m is saved

## PART 2: Write the controller class for the plugin

Starting from version 2, MIB is utilizing Controller-Model-View architecture (read for example the following article to grab the concept: <http://www.tomdalling.com/blog/software-design/model-view-controller-explained/>).

The actual code of the plugin should be located in the Controller class that is automatically started when a user chooses the corresponding plugin from MIB menu. The GUI of the class is stored in the View class that is automatically generated. The contents of the Model class has handles of all image datasets opened in MIB.

To start the Controller class we can use a template (*mibChildController.m*) located in MIB\Development\ folder.

1. Copy *mibChildController.m* from MIB\Development to MIB\Plugins\Tutorials\GuiTutorial directory
2. Rename *mibChildController.m* to *GuiTutorialController.m*. It is important that the name of controller should be combined from the directory name of the plugin (GuiTutorial) and Controller, otherwise MIB won't be able to start it!
3. Open *GuiTutorialController.m* in Matlab editor.
4. Rename *mibChildController* to *GuiTutorialController* everywhere in *GuiTutorialController.m* function text. You can use Ctrl+F shortcut to start Find & Replace dialog
5. Navigate to `function obj = GuiTutorialController(mibModel)`, which is a constructor for the *GuiTutorialController* class.
6. Replace `guiName = 'mibChildGUI';` with the name of the GUI function of the plugin:  
`guiName = 'GuiTutorialGUI';`

```
function obj = GuiTutorialController(mibModel)
    obj.mibModel = mibModel;    % assign model
    guiName = 'GuiTutorialGUI';
    obj.View = mibChildView(obj, guiName); % initialize the view
```

7. Uncomment code responsible for modification of widget sizes within the plugin and replace `obj.View.handles.text1.FontSize` and `obj.View.handles.text1.FontName` to `obj.View.handles.infoText1.FontSize` and `obj.View.handles.infoText1.FontName`

```
% update font and size
global Font;
if obj.View.handles.infoText1.FontSize ~= Font.FontSize ...
    || ~strcmp(obj.View.handles.infoText1.FontName, Font.FontName)
    mibUpdateFontSize(obj.View.gui, Font);
end
```

Important! All handles of the *GuiTutorialGUI* widgets may be accessed from *GuiTutorialController* as `obj.View.handles.[tag of the widget]`. For example, `obj.View.handles.infoText1.String` -> returns the String contents of the *infoText1* widget of the GUI

In addition, the constructor has a section of listeners:

```
obj.listener{1} = addlistener(obj.mibModel, 'updateGuiWidgets', @(src, evnt)
obj.ViewListner_Callback2(obj, src, evnt));
```

The default listener is signed up for the 'updateGuiWidgets' event of mibModel class. This event is fired when the widgets of the main MIB window are updated. The callback is written in the ViewListner\_Callback2 function in the [methods](#) (Static) section of the class and it starts obj.updateWidgets() function that update the contents of the plugin GUI window, so that the plugin window is synchronized with the dataset shown in the Image View panel of MIB.

The full list of mibModel class events can be found in the MIB\Classes\@mibModel\mibModel.m file.

*Additional programming tip:*

If we want to update the contents of the plugin GUI after change of slices in the main MIB window, we have to add another listener:

```
obj.listener{2} = addlistener(obj.mibModel, 'changeSlice', @(src, evnt)
obj.ViewListner_Callback2(obj, src, evnt));
```

and add an additional case statement to ViewListner\_Callback2(obj, src, evnt)

```
function ViewListner_Callback2(obj, src, evnt)
    switch evnt.EventName
        case {'updateGuiWidgets'}
            obj.updateWidgets();
        case {'changeSlice'}
            obj.updateWidgets();
    end
end
```

Add contents of convertPopup and set cropRadio value to 1

```
% populate the contents of convertPopup
obj.View.handles.convertPopup.String = {'uint8', 'uint16'};
obj.View.handles.cropRadio.Value = 1;
```

All additional initialization of the plugin is done in `function updateWidgets(obj)`. However, before continuing with it, lets modify code in GuiTutorialGUI.m

### PART 3: Required modification of GuiTutorialGUI.m

In order to be used with the controller the user interface code (GuiTutorialGUI.m) should be a bit modified.

1. add the following code in the beginning of GuiTutorialGUI\_OpeningFcn(hObject, eventdata, handles, varargin). This code assigns a handle of the controller window to *handles.winController* variable of *GuiTutorialGUI.m*

```
% obtain controller
handles.winController = varargin{1};
```

2. Add the closing plugin function (handles.winController.closeWindow();) to cancelBtn\_Callback. Press of the Cancel button triggers this callback, which in turn starts the closeWindow function of the Controller class

```
% --- Executes on button press in cancelBtn.
function cancelBtn_Callback(hObject, eventdata, handles)
% hObject    handle to cancelBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.winController.closeWindow();
```

3. Add the closing plugin function (handles.winController.closeWindow();) to GuiTutorialGUI\_CloseRequestFcn. Delete delete(hObject); code. This function is triggered when user press the X button of the plugin window.

```
% --- Executes when user attempts to close GuiTutorialGUI.
function GuiTutorialGUI_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to GuiTutorialGUI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.winController.closeWindow();
```

After this step we can already test the plugin from *MIB->Menu->Plugins->Tutorials->GuiTutorial* (remember to save both files: *GuiTutorialGUI.m* and *GuiTutorialController.m*).

## PART 4: Writing updateWidgets function of GuiTutorialController

It is recommended to have updateWidgets function in any plugin. This function may be triggered by the `updateGuiWidgets` listener and updates the contents of the plugin window.

### 1. Get current dimensions of the dataset and update corresponding fields in the GUI

```
% get information about dataset dimensions in pixels
options.blockModeSwitch=0;
[height, width, colors, depth, time] =
    obj.mibModel.I{obj.mibModel.Id}.getDatasetDimensions('image', 4, 0, options);
% force to switch off the BlockMode to make sure that dimensions of the whole dataset
% will be returned
% 'image' - indicates type of the layer to get dimensions
% (other options: 'model','mask','selection')
% 4 - forces to return dimensions of the dataset in the original (XY) orientation.
% 0 - requires to return number of all color channels of the dataset
```

The images stored in MIB can be accessed either using methods of the `mibModel` class as:

```
% get index of the currently shown dataset
currentDatasetIndex = obj.mibModel.Id;
or using methods of mibImage class called as obj.mibModel.I{obj.mibModel.Id}.[method
name], for example:
% get currently shown slice
img = obj.mibModel.I{obj.mibModel.Id}.getData2D('image');
```

The returned `img` – is a cell array, access the image as `img{1}`. For example, `imshow(img{1})`;

*Please check programming help MIB->Menu->Help->Class reference  
for description of methods and functions*

### 2. Use the obtained numbers to populate the edit boxes

```
% populate the edit boxes
obj.View.handles.xMinEdit.String = '1';
obj.View.handles.yMinEdit.String = '1';
% width of the dataset
obj.View.handles.widthEdit.String = num2str(width);
% height of the dataset
obj.View.handles.heightEdit.String = num2str(height);
```

### 3. and the color channel popup menu

```
% populate colorPopup
colorsList = cell([colors, 1]); % allocate space
% generate cell array with color names
for i=1:colors
    colorsList{i} = sprintf('Channel %d', i);
end
% assing the cell array with color names to colorPopup
obj.View.handles.colorPopup.String = colorsList;
```



## PART 5: Writing functions of GuiTutorialGUI that do not require access to mibModel class

Functions of the GUI window that do not require access to the mibModel/mibImage classes can be written directly in the GuiTutorialGUI.m.

1. Navigate to *buttonGroup\_SelectionChangedFcn* and modify it such that it displays only the widgets that are used during each of the actions.

```
function buttonGroup_SelectionChangedFcn(hObject, eventdata, handles)
```

```
% hObject    handle to the selected object in buttonGroup
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% enable all widgets
```

```
handles.xMinEdit.Enable = 'on';
```

```
handles.yMinEdit.Enable = 'on';
```

```
handles.widthEdit.Enable = 'on';
```

```
handles.heightEdit.Enable = 'on';
```

```
handles.convertPopup.Enable = 'on';
```

```
handles.colorPopup.Enable = 'on';
```

```
% switch off widgets that are not used in each of the following modes
```

```
switch hObject.Tag
```

```
    case 'cropRadio' % crop mode
        handles.convertPopup.Enable = 'off';
        handles.colorPopup.Enable = 'off';
```

```
    case 'resizeRadio' % resize mode
        handles.xMinEdit.Enable = 'off';
        handles.yMinEdit.Enable = 'off';
        handles.convertPopup.Enable = 'off';
        handles.colorPopup.Enable = 'off';
```

```
    case 'convertRadio' % convert mode
        handles.xMinEdit.Enable = 'off';
        handles.yMinEdit.Enable = 'off';
        handles.widthEdit.Enable = 'off';
        handles.heightEdit.Enable = 'off';
        handles.colorPopup.Enable = 'off';
```

```
    case 'invertRadio' % invert mode
        handles.xMinEdit.Enable = 'off';
        handles.yMinEdit.Enable = 'off';
        handles.widthEdit.Enable = 'off';
        handles.heightEdit.Enable = 'off';
        handles.convertPopup.Enable = 'off';
```

```
end
```

2. Navigate to *continueBtn\_Callback* and modify it to start the proper function, depending on user choice. *We still have to write the actual functions later.*

```
% --- Executes on button press in continueBtn.
function continueBtn_Callback(hObject, eventdata, handles)
% hObject    handle to continueBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% triggers the main function that does one of the following actions
if handles.cropRadio.Value == 1
    % do cropping
    handles.winController.cropDataset();
elseif handles.resizeRadio.Value == 1
    % do resizing
    handles.winController.resizeDataset();
elseif handles.convertRadio.Value == 1
    % do class conversion
    handles.winController.convertDataset();
elseif handles.invertRadio.Value == 1
    % do invert
    handles.winController.invertDataset();
end
```

## PART 6: Write cropDataset function of GuiTutorialController class

```
function cropDataset(obj)
% function cropDataset(obj)
% crop dataset

% get new dimensions
x1 = str2double(obj.View.handles.xMinEdit.String);
y1 = str2double(obj.View.handles.yMinEdit.String);
width1 = str2double(obj.View.handles.widthEdit.String);
height1 = str2double(obj.View.handles.heightEdit.String);

% get information about the current dataset dimensions in pixels
options.blockModeSwitch=0;
[height, width, colors, depth, time] =
obj.mibModel.I{obj.mibModel.Id}.getDatasetDimensions('image', 4, 0, options);

% check dimensions
if x1 < 1 || y1 < 1 || x1+width1-1 > width || y1+height1-1 > height
    % cancel if dimensions are wrong
    errordlg('Please check dimensions for cropping!', 'Wrong dimensions');
    return
end

% create a waitbar
wb = waitbar(0, 'Please wait...');

% get the whole dataset, use the getData2D, getData3D,
% getData4D of mibModel class to obtain 2D, 3D, or 4D dataset
% the 'image' parameter defines that the image is required
% The function returns a cell with the image
% because complete dataset should be cropped, the blockModeSwitch is forced
% to be 0
img = obj.mibModel.getData4D('image', NaN, 0, options);

% crop the dataset
img{1} = img{1}(y1:y1+height-1, x1:x1+width1-1, :, :, :);

% update the image layer
obj.mibModel.setData4D('image', img, NaN, 0, options);

waitbar(0.5, wb); % update waitbar

% in addition it is needed to resize the other layers
% (Selection, Mask and Model).
% it can be done in a single step when the 63-material type of the model
% is used (obj.mibModel.I{obj.mibModel.Id}.modelType==63) or one by one
% when it is 255-material type
(obj.mibModel.I{obj.mibModel.Id}.modelType~=63).

if obj.mibModel.I{obj.mibModel.Id}.modelType==63
    list = {'everything'};
else
    list = {'selection', 'model', 'mask'};
end
```

```

for layer = 1:numel(list)
    if strcmp(list{layer}, 'mask') && ...
        obj.mibModel.I{obj.mibModel.Id}.maskExist == 0
        % skip when no mask
        continue;
    end
    if strcmp(list{layer}, 'model') && ...
        obj.mibModel.I{obj.mibModel.Id}.modelExist == 0
        % skip when no model
        continue;
    end
    img = obj.mibModel.getData4D(list{layer}, NaN, NaN, options);
    img{1} = img{1}(y1:y1+height-1, x1:x1+width-1, :, :);
    % update the layers
    obj.mibModel.setData4D(list{layer}, img, NaN, NaN, options);

    waitbar(0.5+.1*layer, wb); % update waitbar
end

% update the log text available from MIB->Path panel->Log
log_text = sprintf('Crop: [x1 x2 y1 y2]: %d %d %d %d', ...
    x1, x1+width-1, y1, y1+height-1);
obj.mibModel.I{obj.mibModel.Id}.updateImgInfo(log_text);

% during the crop the BoundingBox is changed, so it should be fixed.
% calculate the shift of the coordinates in the dataset units
% xyzShift = [x1-1 y1-1 0];

% shift of the bounding box in X
xyzShift(1) = (x1-1)*obj.mibModel.I{obj.mibModel.Id}.pixSize.x;
% shift of the bounding box in Y
xyzShift(2) = (y1-1)*obj.mibModel.I{obj.mibModel.Id}.pixSize.y;
% shift of the bounding box in Z
xyzShift(3) = 0;
% update BoundingBox Coordinates
obj.mibModel.I{obj.mibModel.Id}.updateBoundingBox(NaN, xyzShift);
waitbar(1, wb); % update waitbar

% notify MIB that the dataset has been updated
% see more events in the events section of mibModel.m
notify(obj.mibModel, 'newDataset');
% ask MIB to redraw the dataset
notify(obj.mibModel, 'plotImage');

delete(wb); % delete waitbar
end

```

## PART 7: Write resizeDataset function of GuiTutorialController class

```
function resizeDataset(obj)
% function resizeDataset(obj)
% resize the current dataset

% get new dimensions
width1 = str2double(obj.View.handles.widthEdit.String);
height1 = str2double(obj.View.handles.heightEdit.String);

% check dimensions
if width1 < 1 || height1 < 1
    % cancel if dimensions are wrong
    errordlg('Please check dimensions for resizing!', 'Wrong dimensions');
    return
end
wb = waitbar(0, 'Please wait...'); % create a waitbar

% get the whole dataset
% the 'image' parameter defines that the image is required
% The function returns a cell with the image
% because complete dataset should be resized, the blockModeSwitch is forced
to be 0
options.blockModeSwitch = 0;
img = obj.mibModel.getData4D('image', NaN, 0, options);
% get the current dataset dimensions
[height, width] =
obj.mibModel.I{obj.mibModel.Id}.getDatasetDimensions('image', 4, 0, options);

% allocate memory for the output dataset
imgOut = zeros([height1, width1, size(img{1}, 3), size(img{1}, 4),
size(img{1}, 5)], class(img{1}));

% loop the time dimension
for t=1:size(img{1},5)
    % loop the depth dimension
    for slice = 1:size(img{1},4)
        imgOut(:,:,,slice,t) = ...
            imresize(img{1}(:,:,,slice,t),[height1, width1],'bicubic');
    end
end
waitbar(0.5, wb); % update waitbar

% update the image
obj.mibModel.setData4D('image', imgOut, NaN, 0, options);

% in addition it is needed to resize the other layers
% (Selection, Mask and Model).
% it can be done in a single step when the 63-material type of the model
% is used (obj.mibModel.I{obj.mibModel.Id}.modelType==63) or one by one
% when it is 255-material type
(obj.mibModel.I{obj.mibModel.Id}.modelType~=63).
```

```

if obj.mibModel.I{obj.mibModel.Id}.modelType==63
    list = {'everything'};
else
    list = {'selection', 'model', 'mask'};
end
for layer = 1:numel(list)
    if strcmp(list{layer},'mask') && ...
        obj.mibModel.I{obj.mibModel.Id}.maskExist == 0
        continue; % skip when no mask
    end
    if strcmp(list{layer},'model') && ...
        obj.mibModel.I{obj.mibModel.Id}.modelExist == 0
        continue; % skip when no model

    end
    % get the dataset
    img = obj.mibModel.getData4D(list{layer}, NaN, NaN, options);

    % allocate memory for the output dataset
    imgOut = zeros([height1, width1, size(img{1},3), size(img{1},4)],
class(img{1}));
    for t=1:size(img{1},4) % loop the time dimension
        for slice = 1:size(img{1},3) % loop the depth dimension
            % it is important to use nearest resizing method for these layers
            imgOut(:,:,slice,t) = ...
                imresize(img{1}(:,:,slice,t),[height1, width1],'nearest');
        end
    end
    % update the layers
    obj.mibModel.setData4D(list{layer}, img, NaN, NaN, options);
    waitbar(0.5+.1*layer, wb); % update waitbar
end

% generate log text with description of the performed actions, the log can be
accessed with the Log button in the Path panel
log_text = sprintf('Resized to (height x width) %d x %d', height1, width1);
obj.mibModel.I{obj.mibModel.Id}.updateImgInfo(log_text);

% update pixel size for the x and y. The z was not changed
% get current pixel size
pixSize = obj.mibModel.I{obj.mibModel.Id}.pixSize;
pixSize.x = ...
    obj.mibModel.I{obj.mibModel.Id}.pixSize.x/width1*width;
pixSize.y = ...
    obj.mibModel.I{obj.mibModel.Id}.pixSize.y/height1*height;
% update pixels size for the dataset
obj.mibModel.updateParameters(pixSize)
waitbar(1, wb); % update waitbar
% notify MIB that the dataset has been updated
% see more events in the events section of mibModel.m
notify(obj.mibModel, 'newDataset');
% ask MIB to redraw the dataset
notify(obj.mibModel, 'plotImage');

delete(wb); % delete waitbar
end

```

## PART 8: Write convertDataset function of GuiTutorialController class

### Convert dataset between 8 bit and 16 bit datasets

```
function convertDataset(obj)
% function convertDataset(obj)
% convert dataset to a different image class

% get class to convert image to
List = obj.View.handles.convertPopup.String;
val = obj.View.handles.convertPopup.Value;
convertTo = List{val};

% get the dataset
% the 'image' parameter defines that the image is required
% The function returns a cell with the image
% because complete dataset should change its clas, the blockModeSwitch is
forced to be 0
options.blockModeSwitch = 0;
img = obj.mibModel.getData4D('image', 4, 0, options);
classFrom = class(img{1}); % get current image class

% check whether the destination is the same class as the
% current
if strcmp(classFrom, convertTo)
    warndlg(sprintf('The current dataset is already %s class!', convertTo));
    return;
end
wb = waitbar(0, 'Please wait...');
if strcmp(convertTo, 'uint16') % convert to uint16 class
    % calculate stretching coefficient
    coef = double(intmax('uint16')) / double(intmax(class(img{1})));

    % convert stretch dataset to uint16
    img{1} = uint16(img{1})*coef;

else % convert to uint8 class
    % calculate stretching coefficient
    coef = double(intmax('uint8')) / double(intmax(class(img{1})));

    % convert stretch dataset to uint16
    img{1} = uint8(img{1}*coef);
end
waitbar(0.5, wb);
% update dataset in MIB
obj.mibModel.setData4D('image', img, 4, 0, options);

% when the image class has been changed it is important to update the
handles.h.Img{handles.h.Id}.I.viewPort structure.
obj.mibModel.I{obj.mibModel.Id}.updateDisplayParameters();
```

```

% generate log text with description of the performed actions, the log can be
accessed with the Log button in the Path
% panel
log_text = sprintf('Converted from %s to %s', classFrom, class(img{1}));
obj.mibModel.I{obj.mibModel.Id}.updateImgInfo(log_text);
waitbar(1, wb);
% ask MIB to update own widgets, because it is required to
% update a checkbox in MIB->Menu->Image->Mode
notify(obj.mibModel, 'updateId');

% ask MIB to update the shown image
notify(obj.mibModel, 'plotImage');
delete(wb);      % delete waitbar
end

```

## PART 9: Write invertDataset function of GuiTutorialController class

The invert function will invert the selected in *colorPopup* channel. In addition, it will also be compatible with the ROIs. If the ROIs are shown the inverted area will only be inside the shown ROI area.

```

function invertDataset(obj)
% function invertDataset(obj)
% invert the dataset

wb = waitbar(0, 'Please wait...');

% get the color channel to invert
colCh = obj.View.handles.colorPopup.Value;

% in this function we will use getData2D function to show how
% to minimize memory consumption

% get the current dataset dimensions
[height, width, colors, depth, time] =
obj.mibModel.I{obj.mibModel.Id}.getDatasetDimensions('image');

% specify the roiId option: when ROIs are shown the dataset
% will be not the full dataset, but only the areas that are
% inside the ROI areas
options.roiId = [];

% do backup if time dimension is 1
% after backup the previous state can be restored using the
% Ctrl+Z shortcut
if time == 1
    storeOptions.roiId = [];      % backup only the ROI areas
    obj.mibModel.mibDoBackup('image', 1, storeOptions);
end

```



```

index = 1; % define dataset counter for the waitbar
maxIndex = depth*time; % find the maximal number of images to process
% loop across the time dimension
for t=1:time
    % loop across the depth dimension
    for z=1:depth
        % get 2D slice
        img = obj.mibModel.getData2D('image', z, NaN, colCh, options);
        % find maximal point of the dataset
        maxInt = intmax(class(img{1}));
        % loop across the ROIs
        for i=1:numel(img)
            % invert the image
            img{i} = maxInt - img{i};
        end
        obj.mibModel.setData2D('image', img, z, NaN, colCh, options);
        waitbar(index/maxIndex, wb);
        index = index + 1;
    end
end

% update the log text
log_text = 'Invert image';
obj.mibModel.I{obj.mibModel.Id}.updateImgInfo(log_text);
% ask MIB to update the shown image
notify(obj.mibModel, 'plotImage');
delete(wb); % delete waitbar
end

```

## PART 10: Use the plugin

1. Save both GuiTutorialController and GuiTutorialGUI
2. Restart MIB and start the plugin in MIB->Menu->Plugins->Tutorials->GuiTutorial