How to make an own plugin with GUI for Microscopy Image Browser

(*Advanced case, MIB version 1.1*)

Ilya Belevich and Darshan Kumar

This tutorial demonstrates how to write a GUI (Graphical User Interface) based plugin and connect it to Microscopy Image Browser (MIB, im_browser). It is recommended to check first the first tutorial "Add an own function to Microscopy Image Browser" before starting this one.

http://mib.helsinki.fi/tutorials_programming.html

Task: write a GUI add-on that allows crop, resize or convert images to the *uint16* class.

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

```
im_browser\Plugins\Tutorials\guiTutorial\
```

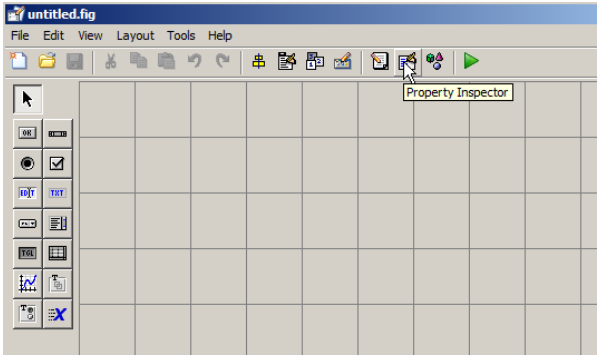
1. Start *GUIDE* (type 'guide' in the Matlab command window).

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

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

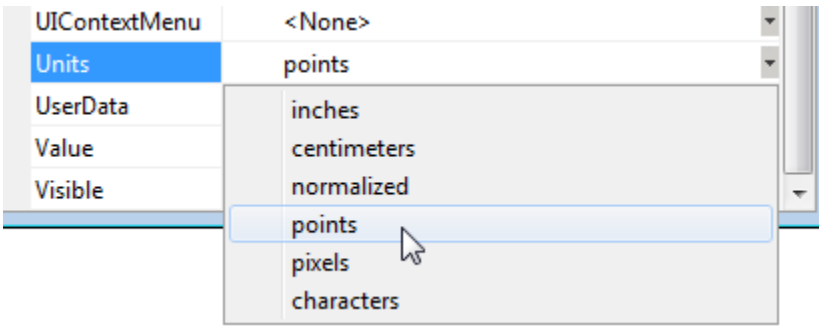
Few notes about design:

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

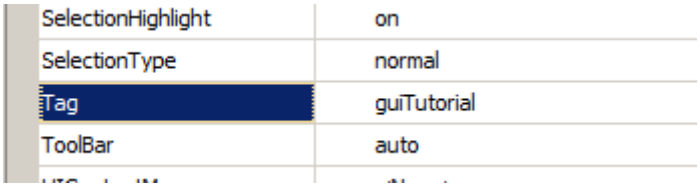


- Important!

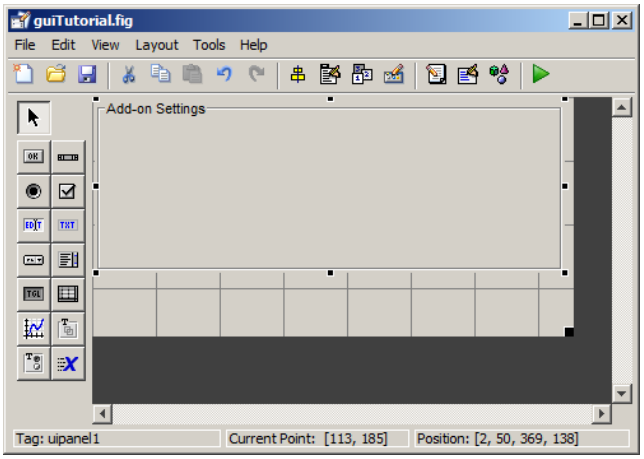
For proper sizing of widgets across different platforms set units for all placed widgets to points!



3. Assign a name of the plugin: set the *Tag* of the main figure to *guiTutorial* in the Inspector window

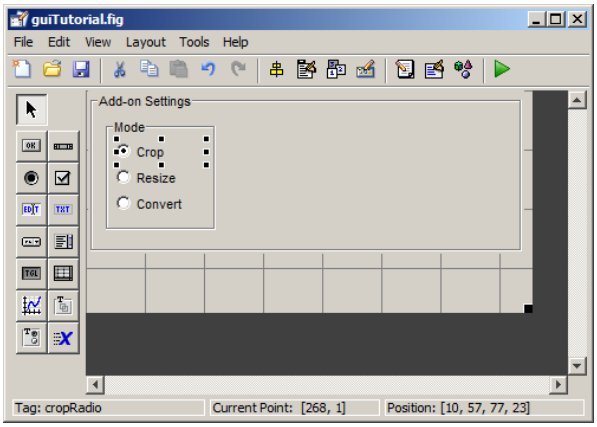


4. Create a panel and add title: 'Add-on Settings' (*Inspector -> Title*).



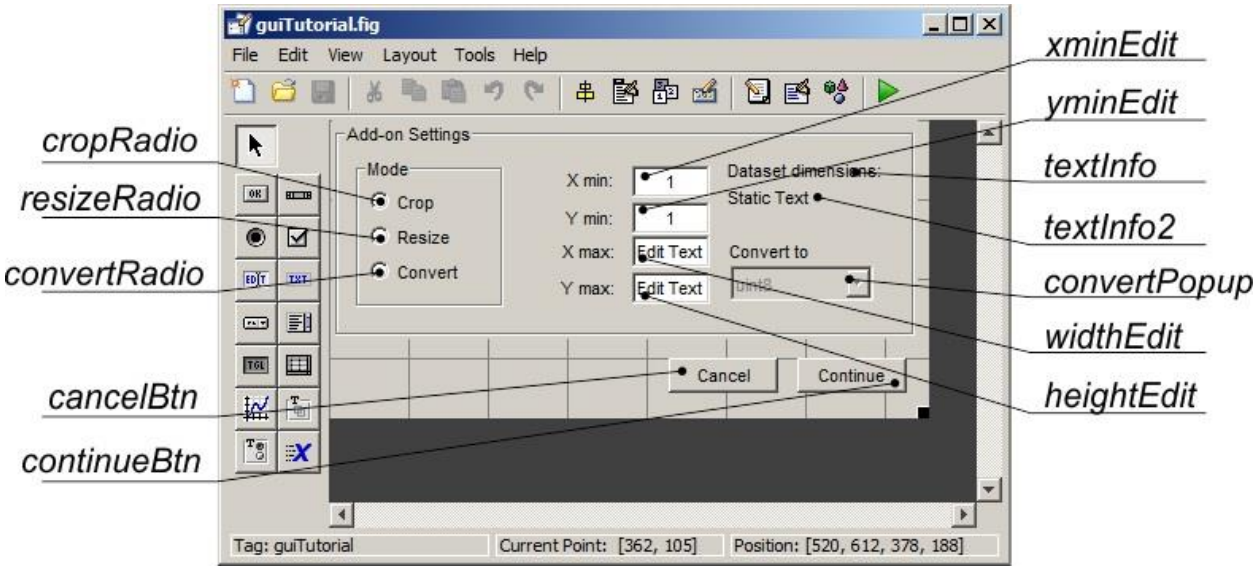
5. Add a Radio Group  and assign its Name to "Mode" in the Inspector window, set its tag to `buttonGroup`.

6. Add to the group 3 radio buttons ("Crop", "Resize", "Convert") tagged (*Inspector->Tag*) as *cropRadio*, *resizeRadio* and *convertRadio* respectively.

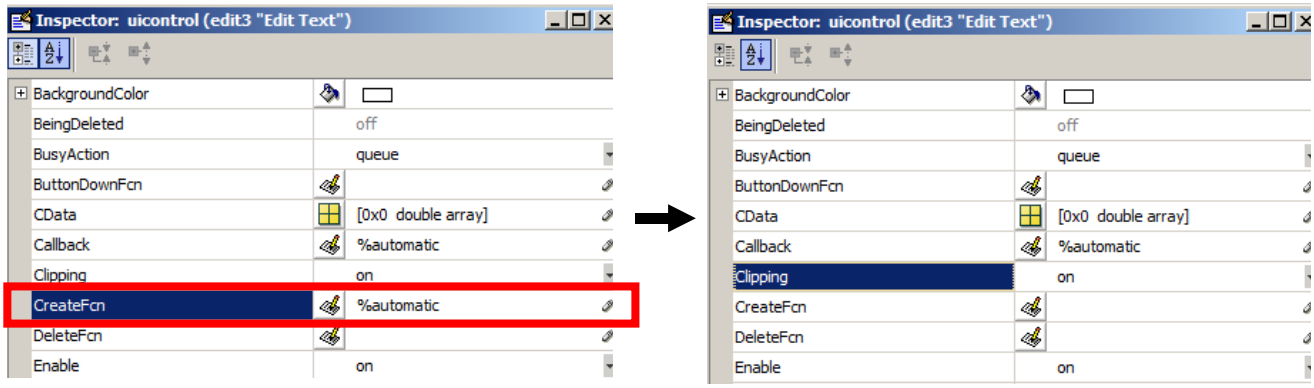


7. Add additional components:

- 4 edit boxes (*xminEdit, yminEdit, widthEdit, heightEdit*)
- a popup-menu (*convertPopup*),
- 2 push buttons (*cancelBtn, continueBtn*)
- few text fields and assign appropriate tags to them as shown in the snapshot below



8. Remove create functions (*Inspector->CreateFcn*) for the popup menus and the edit boxes to make the final code simpler. Click on each of these widgets and delete the *%automatic* from the *CreateFcn* field.



9. Keep callbacks for the *widthEdit* and *heightEdit*; remove callbacks from *xminEdit*, *yminEdit*, and *convertPopup* - delete the "`%automatic`" text from the Callback fields in the Inspector window.

10. For *convertPopup* set *Inspector->Enable->Off* and add *uint8* and *uint16* to its *String* property (*Inspector->String->String button* ).

11. Save figure and matlab code (GUIDE->Menu->Save as…) to the `Plugins\Tutorials\guiTutorial` directory under the im_browser directory.
For example, the full path may look like this:
`C:\Scripts\im_browser\Plugins\Tutorials\guiTutorial\guiTutorial.fig`
`C:\Scripts\im_browser\Plugins\Tutorials\guiTutorial\guiTutorial.m`

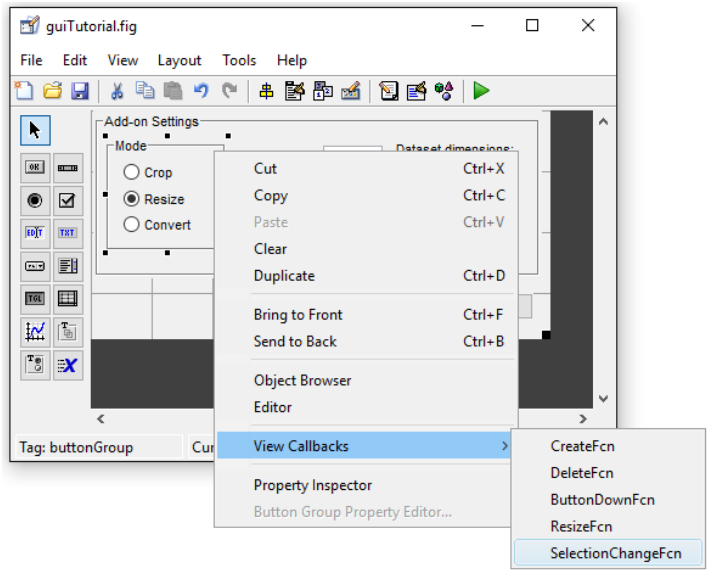
It is important that the directory name should match the name of the main GUI function (*`guiTutorial`* in this example).

12. Create callback for the `buttonGroup` that will be used to define the function mode: Crop, Resize or Convert.

- highlight the Mode group with the left mouse button

- press the right mouse button and select *View Callbacks -> SelectionChangeFcn* to create a common callback for all radio buttons (*cropRadio*, *resizeRadio* and *convertRadio*)

- this will start the Matlab editor



13. Open the *guiTutorial.m* in Matlab editor if the editor was not started in the previous step

14. Find the `guiTutorial_OpeningFcn` (*this function is called during initialization of GUI*) function and add the following code to the beginning of it to do the basic initialization of the plugin.

```
%! get the handle of the main im_browser window
h_im_browser = varargin{3};
%! get the handles structure of the main program (im_browser):
% handles.[tag] - give access to all widgets of the plugin window, while
% handles.h.[tag] - give access to all widgets of im_browser
handles.h = guidata(h_im_browser);

% update font and size of the GUI window, required for better visualization
% of the GUI on different operating systems.
% Remember, that the Units of widgets and the main window should be points!

% NOTE 1: you may need to replace "handles.textInfo" with a text field tag of
your own GUI
if get(handles.textInfo, 'fontsize') ~= ...
      handles.h.preferences.Font.FontSize ||
           ~strcmp(get(handles.textInfo,'fontname'),...
                     handles.h.preferences.Font.FontName)

    % call a function to update font name and size
    % NOTE 2: you have to replace "handles.guiTutorial" with tag of
    % your own GUI
    ib_updateFontSize(handles.guiTutorial, handles.h.preferences.Font);
end

% resize all elements x1.5 times for macOS and x1.25 for Linux
% NOTE 3: you have to replace "handles.guiTutorial" with tag of
% your own GUI
mib_rescaleWidgets(handles.guiTutorial);


% update information about dataset dimensions in pixels
options.blockModeSwitch=0;
[height,width,~,no_stacks, time] = ...
  handles.h.Img{handles.h.Id}.I.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 (not
important in this routine)

% generate information string
infoString = sprintf('%d x %d x %d x %d', width, height, no_stacks, time);
% place the information string to the textInfo2 field
set(handles.textInfo2, 'string', infoString);


% update widthEdit and heightEdit with width and height parameters of the
dataset
set(handles.widthEdit, 'string', num2str(width));
set(handles.heightEdit, 'string', num2str(height));
```

From this moment it is possible to get access to classes of `im_browser` (see more in "Add an own function to Microscopy Image Browser" tutorial and help: *im_browser->Menu->Help->Class reference* or http://mib.helsinki.fi/help/api/classimage_data.html

There following classes are available:

- *handles.h.Img{index}.I* – a class where the images are stored.
- *handles.h.Img{index}.I.hROI* – a subclass with information about regions of interest defined in the ROI panel
- *handles.h.Img{index}.I. hLabels* - a subclass that contains information about annotation labels
- *handles.h.Img{index}.I. hMeasure* - a subclass that contains measurements (angles, radii, length…)
- *handles.h.U* – a class with information of the undo system

The access to the currently selected image is done using `handles.h.Img{handles.h.Id}.I.[functionName]` syntax, where `handles.h.Id` contains the index of the opened dataset from 1 to 8.
For example,

```
[height, width, colors, depth, time] = ...
  handles.h.Img{handles.h.Id}.I.getDatasetDimensions('image', 0, 0, options);
```

15. Find and add the following code to `cancelBtn_Callback` to close the figure after press of the Cancel button.

```
delete(handles.guiTutorial);      % delete the window
```

16. Find and modify *widthEdit_Callback* to preserve aspect ratio when modifying the width of the output image

```
% get dataset dimensions

% force to switch off the BlockMode switch,
% to get dimensions of the whole dataset
options.blockModeSwitch=0;
[height, width, ~, ~] =
handles.h.Img{handles.h.Id}.I.getDatasetDimensions('image', 4, 0, options);

% get entered new width
newWidth = str2double(get(handles.widthEdit,'string'));
% calculate height/width ratio
ratio = height/width;
% calculate new height
newHeight = round(newWidth*ratio);

% update the height edit box to preserve the ratio
set(handles.heightEdit, 'string', num2str(newHeight));
```

17. Find and modify *heightEdit_Callback* to preserve aspect ratio when changing height of the output image.

```
% get dataset dimensions
% force to switch off the BlockMode switch, to get dimensions of the whole
dataset
options.blockModeSwitch=0;
[height, width, ~, ~] =
handles.h.Img{handles.h.Id}.I.getDatasetDimensions('image', 4, NaN, options);

% get entered new height
newHeight = str2double(get(handles.heightEdit,'string'));
% calculate height/width ratio
ratio = height/width;
% calculate new width
newWidth = round(newHeight/ratio);
% update the height edit box to preserve the ratio
set(handles.widthEdit, 'string', num2str(newWidth));
```

18. Modify `buttonGroup_SelectionChangeFcn` that handles callbacks from the radio buttons (`cropRadio, resizeRadio, convertRadio`).

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

% get identifier of the radio button
tag = get(hObject, 'tag');

% disable all elements
set(handles.widthEdit,'enable', 'off');
set(handles.heightEdit,'enable', 'off');
set(handles.xminEdit,'enable', 'off');
set(handles.yminEdit,'enable', 'off');
set(handles.convertPopup,'enable', 'off');

% tweak some of gui to visualize only required elements of GUI
switch tag
    case 'cropRadio'
        set(handles.widthEdit,'enable', 'on');
        set(handles.heightEdit,'enable', 'on');
        set(handles.xminEdit,'enable', 'on');
        set(handles.yminEdit,'enable', 'on');
        set(handles.textWidth, 'String', 'X max:');
        set(handles.textHeight, 'String', 'Y max:');
    case 'resizeRadio'
        set(handles.widthEdit,'enable', 'on');
        set(handles.heightEdit,'enable', 'on');
        set(handles.textWidth, 'String', 'Image width:');
        set(handles.textHeight, 'String', 'Image height:');
    case 'convertRadio'
        set(handles.convertPopup,'enable', 'on');
end
```

19. Find and program the Continue button (continueBtn_Callback)

```
% get parameters from the gui
xmin = str2double(get(handles.xminEdit,'string'));
ymin = str2double(get(handles.yminEdit,'string'));
width = str2double(get(handles.widthEdit,'string'));
height = str2double(get(handles.heightEdit,'string'));
list = get(handles.convertPopup,'string');
listValue = get(handles.convertPopup,'value');
convertTo = list{listValue};

% start functions that perform required actions
% the function can be inside this file of stored in the plugin directory
if get(handles.convertRadio, 'value')   % convert image
    convertImage(handles, convertTo);
elseif get(handles.resizeRadio,'value') % resize image
    resizeImage(handles, width, height);
elseif get(handles.cropRadio, 'value')  % crop image
    cropDataset(handles, xmin, ymin, width, height)
end
```

20. Write *convertImage* function to convert dataset to a different image class

```
function convertImage(handles, convertTo)
% convert dataset to a different image class

% get the whole dataset
% the 'image' parameter defines that the image is required
% The function returns a cell array with the image(s);
% since the whole dataset should be converted, so the blockModeSwitch should
be forced to be 0
options.blockModeSwitch = 0;
img = ib_getDataset('image', handles.h, 4, 0, options);

classFrom = class(img{1});  % get current image class
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

% update dataset in im_browser
ib_setDataset('image', img, handles.h, 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.
handles.h.Img{handles.h.Id}.I.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}));
handles.h.Img{handles.h.Id}.I.updateImgInfo(log_text);

% update widgets in the im_browser GUI
handles.h = updateGuiWidgets(handles.h);

% redraw image in the im_browser axes
handles.h.Img{handles.h.Id}.I.plotImage(handles.h.imageAxes, handles.h, 1);

% close guiTutorial window
delete(handles.guiTutorial);
```

21. Write *resizeImage* function to resize the image dataset

```
function resizeImage(handles, width, height)
% resize dataset and all other layers

% 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 = ib_getDataset('image', handles.h, NaN, 0, options);

% allocate memory for the output dataset
imgOut = zeros([height, width, 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),[height,
width],'bicubic');
    end
end

% update the image, the image should be sent as a cell
ib_setDataset('image', {imgOut}, handles.h, NaN, 0, options);
```

In addition to the Image layer it is needed to resize other layers: Mask, Selection and Model. When the 63 material model is used (*Menu->File->Preferences->Models->Maximal number of materials in model->63*) it is required to resize only one layer (called '*everything*'), but if the model type is 255 all three layers have to be resized.

```
% in addition it is needed to resize the other existing layers:
% Selection, Mask and Model. It can be done in a single step when the uint6 type of
the model is used (handles.h.Img{handles.h.Id}.I.model_type=='uint6') or one by one
when it is uint8 type: handles.h.Img{handles.h.Id}.I.model_type=='uint8'

if strcmp(handles.h.Img{handles.h.Id}.I.model_type, 'uint6')
    % get everything, i.e. it take in one step all three layers
    img = ib_getDataset('everything', handles.h, NaN, NaN, options);
    % allocate memory for the output dataset
    imgOut = zeros([height, width, size(img{1},3),size(img{1},4)],...
                                                        class(img{1}));

    for t=1:size(img{1},4)
        for slice = 1:size(img{1},3)
            % it is important to use nearest resizing method for these layers
            imgOut(:,:,slice,t) = ...
                    imresize(img{1}(:,:,slice,t),[height, width],'nearest');
        end
    end

    % update the service layers, the image should be sended as a cell
    ib_setDataset('everything', {imgOut}, handles.h, NaN, NaN, options);

else   % when the uint8 type of model
    list = {'selection','model','mask'}; % generate list of layers
    for layer = 1:numel(list)
        if strcmp(list{layer},'mask') && handles.h.Img{handles.h.Id}.I.maskExist == 0;
            % skip when no mask
            continue;
        end
        if strcmp(list{layer},'model') && ...
                                handles.h.Img{handles.h.Id}.I.modelExist == 0;
            % skip when no model
            continue;
        end

        % do resizing
        img = ib_getDataset(list{layer}, handles.h, NaN, NaN, options);

        % allocate memory for the output dataset
        imgOut = zeros([height, width, size(img{1},3),size(img{1},4)], 'uint8');

        for t=1:size(img{1},4)
            for slice = 1:size(img{1},3)
                % it is important to use nearest resizing method for these layers
                imgOut(:,:,slice,t) = ...
                            imresize(img{1}(:,:,slice,t),[height, width],'nearest');
            end
        end
        % update the layer, the image should be sended as a cell
        ib_setDataset(list{layer}, {imgOut}, handles.h, NaN, NaN, options);
    end
end
```

Finish the function

```
% 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', height, width);
handles.h.Img{handles.h.Id}.I.updateImgInfo(log_text);

% update pixel size for the x and y. The z was not changed
handles.h.Img{handles.h.Id}.I.pixSize.x = ...
    handles.h.Img{handles.h.Id}.I.pixSize.x/height*size(img{1},1);
handles.h.Img{handles.h.Id}.I.pixSize.y = ...
    handles.h.Img{handles.h.Id}.I.pixSize.y/height*size(img{1},1);
handles.h.Img{handles.h.Id}.I.pixSize.z = ...
    handles.h.Img{handles.h.Id}.I.pixSize.z;

% update img_info XResolution and YResolution
resolution = ib_calculateResolution(handles.h.Img{handles.h.Id}.I.pixSize);
handles.h.Img{handles.h.Id}.I.img_info('XResolution') = resolution(1);
handles.h.Img{handles.h.Id}.I.img_info('YResolution') = resolution(2);
handles.h.Img{handles.h.Id}.I.img_info('ResolutionUnit') = 'Inch';

% update widgets in the im_browser GUI
handles.h = updateGuiWidgets(handles.h);

% because the dataset was resized the image axes should be updated!
handles.h = handles.h.Img{handles.h.Id}.I.updateAxesLimits(handles.h,
'resize');

% redraw image in the im_browser axes
handles.h.Img{handles.h.Id}.I.plotImage(handles.h.imageAxes, handles.h, 1);

% close guiTutorial window
delete(handles.guiTutorial);
```

22. Write *cropImage* function to crop dataset. Similar to *resizeImage*, we start with the crop of the Image layer

```
function cropDataset(handles, xmin, ymin, xmax, ymax)
% crop the dataset in xy dimension keeping the number of stacks intact
% 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 cropped, the blockModeSwitch is forced to be 0
options.blockModeSwitch = 0;
img = ib_getDataset('image', handles.h, NaN, 0, options);

% crop the dataset
img{1} = img{1}(ymin:ymax, xmin:xmax, : , :, :);

% update the image layer
ib_setDataset('image', img, handles.h, NaN, 0, options);
```

After that crop all other layers

```
if strcmp(handles.h.Img{handles.h.Id}.I.model_type, 'uint6')
    % get everything, i.e. it take in one step all three layers
    % (selection, mask, model)
    img = ib_getDataset('everything', handles.h, NaN, NaN, options);
    img{1} = img{1}(ymin:ymax, xmin:xmax, :, :);
    % update the layers
    ib_setDataset('everything', img, handles.h, NaN, NaN, options);
else % when the uint8 type of model
    list = {'selection','model','mask'};
    for layer = 1:numel(list)
        if strcmp(list{layer},'mask') && handles.h.Img{handles.h.Id}.I.maskExist == 0;
            % skip when no mask
            continue;
        end
        if strcmp(list{layer},'model') && ...
                          handles.h.Img{handles.h.Id}.I.modelExist == 0;
            % skip when no model
            continue;
        end
        % do crop
        img = ib_getDataset(list{layer}, handles.h, NaN, NaN, options);
        img{1} = img{1}(ymin:ymax, xmin:xmax, :, :);
        % update the layer, the image should be sended as a cell
        ib_setDataset(list{layer}, img, handles.h, NaN, NaN, options);
    end
end

% update the log text
log_text = sprintf('Crop: [x1 x2 y1 y2]: %d %d %d %d', ...
                                        xmin, xmax, ymin, ymax);
handles.h.Img{handles.h.Id}.I.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 = [xmin-1 ymin-1 0];

% shift of the bounding box in X
xyzShift(1) = (xmin-1)*handles.h.Img{handles.h.Id}.I.pixSize.x;
% shift of the bounding box in Y
xyzShift(2) = (ymin-1)*handles.h.Img{handles.h.Id}.I.pixSize.y;
% shift of the bounding box in Z
xyzShift(3) = 0;
% update BoundingBox Coordinates
handles.h.Img{handles.h.Id}.I.updateBoundingBox(NaN, xyzShift);

% because the dataset was resized the image axes should be updated!
handles.h = ...
       handles.h.Img{handles.h.Id}.I.updateAxesLimits(handles.h, 'resize');

% redraw image in the im_browser axes
handles.h.Img{handles.h.Id}.I.plotImage(handles.h.imageAxes, handles.h, 1);

% close guiTutorial window
delete(handles.guiTutorial);
```

23. Save (overwrite the existing file) function to the
`Plugins\Tutorials\guiTutorial\guiTutorial.m`

24. Re-start *im_browser*.
Now the GUI function may be started from the menu:
im_browser->Menu->Plugins->Tutorials-> guiTutorial