CellProfiler Help: DeveloperInfo

Programming Notes for CellProfiler Developer's version

*** INTRODUCTION ***

You can write your own modules, image tools, and data tools for CellProfiler - the easiest way is to modify an existing one. CellProfiler is modular: every module, image tool, and data tool is a single MATLAB m-file (extension = .m). Upon startup, CellProfiler scans its Modules, DataTools, and ImageTools folders looking for files. Simply put your new file in the proper folder and it will appear in the proper place. They are automatically categorized, their help extracted, etc.

If you have never tried computer programming or have not used MATLAB, please do give it a try. Many beginners find this language easy to learn and the code for CellProfiler is heavily documented so that you can understand what each line does. It was designed so that biologists without programming experience could adapt it.

*** HELP SECTIONS AT THE BEGINNING OF EACH MODULE AND TOOL ***

The first unbroken block of lines will be extracted as help by CellProfiler's 'Help for this analysis module' button, Help for image tools and data tools (Help menu in the main CellProfiler window) as well as MATLAB's built in 'help' and 'doc' functions at the command line. It will also be used to automatically generate a pdf manual page for the module. An example image demonstrating the function of the module can also be saved in tif format, using the same name as the module, and it will automatically be included in the pdf manual page as well. Follow the convention of: Help for the XX module, Category (use an exact match of one of the categories so your module appears in the proper place in the "Add module" window), Short description, purpose of the module, description of the settings and acceptable range for each, how it works (technical description), and See also NameOfModule. The license/author information should be separated from the help lines with a blank line so that it does not show up in the help displays.

*** SETTINGS (CALLED 'VARIABLES' IN THE CODE) ***

Variables are automatically extracted from lines in a commented section near the beginning of each module. Even though they look like comments they are critical for the functioning of the code. The syntax here is critical - indenting lines or changing the spaces before and after the equals sign will affect the ability of the variables to be read properly.

* The '%textVAR' lines contain the variable descriptions which are

text will wrap appropriately so it can be as long as desired, but it must be kept on a single line in the m-file (do not allow it to wrap).

* Whether the variable is entered into an edit box, chosen from a popup menu, or selected using browse buttons is determined by %inputtypeVAR

lines and the %textVAR lines. The options are:
- edit box (omit any %inputtypeVAR line for that variable number and use a %defaultVAR line to specify what text will appear in the box when the user first loads the module)
- popup menu (use %inputtypeVAR = popupmenu and then use %choiceVAR lines, in the order you want them to appear, for each option that should appear in the popup menu)
- popupmenu custom (this allows the user to choose from choices but also to have the option of typing in a custom entry. Use %inputtypeVAR = popupmenu custom and then use %choiceVAR lines, in the order you want them to appear, for each option that should appear in the popup menu)
- pathname box + browse button (omit the %inputtypeVAR line and instead use %pathnametextVAR - the default shown in the edit box will be a period; this default is currently not alterable)
- filename box + browse button (omit the %inputtypeVAR line and instead use %filenametextVAR - the default shown in the edit box will be the text "Do not use"; this default is currently not alterable)

* The %infotypeVAR lines specify the group that a particular entry will belong to. You will notice that many entries that the user types into the main window of CellProfiler are then available in popup menus in other modules. This works by classifying certain types of variable entries as follows:
- imagegroup indep: the user's entry will be added to the imagegroup, and will therefore appear in the list of selectable images for variables whose type is 'imagegroup'. Usually used in combination with an edit box; i.e. no %inputtype line.
- imagegroup: will display the user's image entries. Usually used in combination with a popupmenu.
- objectgroup indep and objectgroup: Same idea as imagegroup, for passing along object names.
- outlinegroup indep and outlinegroup: Same idea as imagegroup, for passing along outline names.
- datagroup indep and datagroup: Same idea as imagegroup, for passing along text/data names.
- gridgroup indep and gridgroup: Same idea as imagegroup, for passing along grid names.

* The line of actual code within each group of variable lines is what actually extracts the value that the user has entered in the main window of CellProfiler (which is stored in the handles structure) and saves it as a variable in the workspace of this module with a meaningful name.

* For CellProfiler to load modules and pipelines correctly, the order of variable information should be as follows:
%textVAR01 = Whatever text description you want to appear
%defaultVAR01 = Whatever text you want to appear
(OR, %choiceVAR01 = Whatever text)
%infotypeVAR01 = imagegroup indep
BlaBla = char(handles.Settings.VariableValues{CurrentModuleNum,1});
%inputtypeVAR01 = popupmenu
  For cases in which the variable input is optional or your module should ignore the contents of the variable box, the standard placeholder text is "Do not use." Please follow this naming convention whenever new modules

are created or modified.
In particular, when the input type is "popupmenu custom", the
choiceVAR01 line should be after textVAR01. This order is necessary
because the textVAR01 creates a VariableBox associated with a variable
number. Also, the defaultVAR01 value will inadvertently overwrite saved
settings when loading a saved pipeline if it is located after
infotypeVAR01 or inputtypeVAR01.

When loading the settings of pipeline modules, CellProfiler tries to find
      handles.Settings.VariableValues{ModuleNums,i} from the list of
      handles.VariableBox{ModuleNums}(i),
for example,
      the pipeline-specified 'Gaussian Filter' from the list of
      available Smoothing methods in the loaded module.
It is searched and set in CellProfiler.m, exactly starting
with this line of code:
      PPos = find(strcmp(handles.Settings.VariableValues{ModuleNums,i},OptList));
You may want to add your own action code here when a certain setting is
found in a loaded module.

* CellProfiler uses VariableRevisionNumbers to help programmers notify
users when something significant has changed about the variables.
For example, if you have switched the position of two variables,
loading a pipeline made with the old version of the module will not
behave as expected when using the new version of the module, because
the settings (variables) will be mixed up. The line should use this
syntax:
%%%VariableRevisionNumber = 1
   If the module does not have this line, the VariableRevisionNumber is
assumed to be 0. This number need only be incremented when a change made
to the modules will affect a user's previously saved settings. There is a
revision number at the end of the license info at the top of the m-file
for our source-control revisions - this revision number does not affect
the user's previously saved settings files and you can ignore it.
However, a line with "% $Revision: 5791 $" should be added to any new
function, so that the version-control system will find and update the
number upon new commits.

*** STORING AND RETRIEVING DATA: THE HANDLES STRUCTURE ***

In CellProfiler (and MATLAB in general), each independent function
(module) has its own workspace and is not able to 'see' variables
produced by other modules. For data or images to be shared from one
module to the next, they must be saved to what is called the 'handles
structure'. This is a variable, whose class is 'structure', and whose
name is handles. The contents of the handles structure can be printed out
at the command line of MATLAB using the Tech Diagnosis button and typing
"handles" (no quotes). The only variables present in the *main* handles
structure are handles to figures and GUI elements. Everything else should
be saved in one of the following substructures:

handles.Settings:
Everything in handles.Settings is stored when the user uses File > Save
pipeline, and these data are loaded into CellProfiler when the user uses
File > Load pipeline. This substructure contains all necessary
information to re-create a pipeline, including which modules were used
(including variable revision numbers), their settings (variables), and
the pixel size. Fields currently in handles.Settings: PixelSize,
VariableValues, NumbersOfVariables, VariableInfoTypes,
VariableRevisionNumbers, ModuleNames, SelectedOption.
   *** N.B. handles.Settings.PixelSize is where you should retrieve the
PixelSize if needed, not in handles.Preferences!

handles.Pipeline: This substructure is deleted at the beginning of
the analysis run (see 'Which substructures are deleted prior to an
analysis run?' below). handles.Pipeline is for storing data which
must be retrieved by other modules. This data can be overwritten as
each image cycle is processed, or it can be generated once and then
retrieved during every subsequent image set's processing, or it can
be saved for each image set by saving it according to which image
cycle is being analyzed, depending on how it will be used by other
modules. Example fields in handles.Pipeline: FileListOrigBlue,
PathnameOrigBlue, FilenameOrigBlue, OrigBlue (which contains the
actual image). Whether the handles.Pipeline structure is stored in
the output file or not depends on whether you are in Fast Mode (see
Help > HelpFastMode or File > SetPreferences). See note below for
the FileList..., Pathname..., and Filename... fields.

handles.Current:
This substructure contains information needed for the main CellProfiler
window display and for the various modules and help files to function. It
does not contain any module-specific data (which is in handles.Pipeline).
Example fields in handles.Current: NumberOfModules, StartupDirectory,
DefaultOutputDirectory, DefaultImageDirectory, FilenamesInImageDir,

DataToolsFilenames, DataToolHelp, HelpFilenames, Help, NumberOfImageSets,
SetBeingAnalyzed, SaveOutputHowOften, TimeStarted, CurrentModuleNumber,
FigureNumberForModuleXX.

handles.Preferences:
Everything in handles.Preferences is stored in the file
CellProfilerPreferences.mat when the user uses File > Set Preferences.
These preferences are loaded upon launching CellProfiler, or individual
preferences files can be loaded using File > Load Preferences. Fields in
handles.Preferences: PixelSize, DefaultModuleDirectory,
DefaultOutputDirectory, DefaultImageDirectory, IntensityColorMap,
LabelColorMap, StripPipeline, SkipErrors, FontSize.
   The PixelSize, DefaultImageDirectory, and DefaultOutputDirectory
fields can be changed for the current session by the user using edit
boxes in the main CellProfiler window, which changes their values in
handles.Settings or handles.Current. Therefore:
   *** N.B. handles.Settings.PixelSize is where you should retrieve the
PixelSize if needed, not in handles.Preferences!
   *** N.B. handles.Current.DefaultImageDirectory is where you should
retrieve the DefaultImageDirectory if needed, not in handles.Preferences!
   *** N.B. handles.Current.DefaultOutputDirectory is where you should
retrieve the DefaultOutputDirectory if needed, not in
handles.Preferences!

handles.Measurements:

analyzed and is therefore accessed by the data tools. This substructure
is deleted at the beginning of the analysis run (see 'Which substructures
are deleted prior to an analysis run?' below).
   Note that two types of measurements are typically made: Object
and Image measurements. Object measurements have one number for every
object in the image (e.g. Object Area) and image measurements have one
number for the entire image, which could come from one measurement from
the entire image (e.g. Image TotalIntensity), or which could be an
aggregate measurement based on individual object measurements (e.g. Image
MeanAreaCells). Use the appropriate substructure to ensure that your data
will be extracted properly.
  The relationships between objects can also be defined. For
example, a nucleus might be associated with a particular cytoplasm
and therefore each nucleus has a cytoplasm's number in the nucleus'
measurement field which links the two. Or, for multiple speckles
within a nucleus, each speckle will have a nucleus' number
indicating which nucleus the speckle belongs to (see the Relate
module or Identify Secondary or Tertiary modules). Image
measurements include a few standard fields: ModuleErrorFeatures,
ModuleError, TimeElapsed, FileName_IMAGENAME, and
PathName_IMAGENAME. See note below for fields having to do with
file and path names.

  Measurement storage was overhauled 2008-04-25 such that all modules
that record measurements must use the subfunction CPaddmeasurements. The
usage is:

  handles = CPaddmeasurements(handles,ObjectName,FeatureName,Data);

This will create this data structure:

  handles.Measurements.ObjectName.FeatureName = Data

  where
    -ObjectName is a single string denoting the name of the object, or
      simply "Image" for image measurements
    -FeatureName is a single string, with category and parameters
      (optional) underscored, like this:

                  Category_SpecificFeatureName_Parameters

          * Category = Module name (e.g., AreaShape), or useful category,
              or nothing if there is no appropriate category (e.g., if
              feature name = ObjectCount there is no category).
              - Note: Do not include the word "Measure" when naming.
              - Note: If you create a new category, be sure to add it to
              the list of categories below, as well as in
              CPgetfeaturenamesfromnumbers, and all choiceVAR lists so that
              your new category will be selectable (in the future, this will
              be a drop down menu) for modules that ask the user to
              choose a category.

          * SpecificFeatureName = specific feature recorded by a module
              (e.g., Perimeter). Usually the module recording the
              measurement assigns this name, but a few modules allow the
              user to type in the name of the feature (e.g., the
              CalculateRatios module allows the user to name the ratio).
              - Note: Be sure to list the Specific features measured by
              the module in the Help section. See MeasureObjectAreaShape
              for an example.
          * Parameters (optional) are used for modules that measure the
              same objects in different ways (e.g. the
              MeasureObjectIntensity module can measure intensities for
              Nuclei in two different images, blue and green). Primarily
              used for Channel or scale of Texture. Multiple parameters
              can be separated by underscores.
              (someday, CP will look at upstream modules and make dropdowns)

          Category List:
          These reflect choiceVAR lists in many modules, with their
          necessary extra parameters:
              No extra parameters:
                  AreaShape, Math
              Image:
                  Imageintensity, Granularity, Children, Parent, AreaOccupied
              SizeScale:
                  Neighbors
              SizeScale and Image:
                  Texture and RadialDistribution
              Not to include in choiceVAR lists:
                  Align, Ratio, ClassifyObjects, ClassifyObjsByTwoMeas,
                  ModuleError, Crop (though Crop could be added to the
                  Image group above if needed), DefinedGrid
          When these categories are altered, please update the code in
              CPgetfeaturenamesfromnumbers and any module that uses this
              subfunction.

      Note: CPjoinstrings can be helpful in constructing feature
      names from strings and integers. (If you are just joining
      strings, it is usually more convenient to join them
      directly with ['texture_', stringvariable], etc.)
      Usage: CPjoinstrings('texture',42,'foo') => 'texture_42_foo'

    -Data is either:
      (a) Nx1 vector of numerical data, one number per object where there
      are N objects.
      (b) [], i.e., the empty matrix if the module did not measure any
      objects in this instance. YES, it is very important to pass the
      empty matrix through CPaddmeasurements even if no objects were
      found or measured for a particular image.
      (c) A single string (only makes sense when the ObjectName =
      "Image")
      (d) In the future, we might add the capability to store Nx1
      strings, i.e., one string for every object.

Be sure to consider whether measurements you are storing will overwrite

each other if more than one of the same module is placed in the pipeline. You can differentiate measurements by including something specific in the name (e.g. Intensity modules include the image name (e.g. Blue or Green) in the substructure name). There are also several examples of modules where new measures are appended to the end of an existing substructure (i.e. forming a new column). See Calculate Ratios.

handles.Measurements: Order
Be certain that the order in which measurements are added correspond to the Feature & FeatureNumber in each function's Help section. This FeatureNumber will correspond to the order of CPaddmeasurements statements within each function. In the future, this will be superceded by a more intelligent measurement selection system using context dependent drop-down slectors.

%%%%
Why are file names stored in several places in the handles structure? The Load Images module creates all of the following:
- handles.Pipeline.FileListIMAGENAME
- handles.Pipeline.Pathname
- handles.Pipeline.FilenameIMAGENAME
- handles.Measurements.Image.PathName_IMAGENAME
- handles.Measurements.Image.FileName_IMAGENAME

The primary reason for the fields in the Measurements branch is that it allows the information to be exported easily. However, these fields are also used elsewhere, e.g., the SaveImages module.

The FileList field is mainly useful for movies. For movies, the FileList field has the original name of the movie file and how many frames it contains. The Filenames field has the original movie file name and appends the frame number for every frame in the movie. This allows the names to be used in other modules such as SaveImages, which would otherwise over-write itself on every cycle using the original file name. The FileList location is created at the beginning of the run and contains all the images that will possibly be analyzed, whereas the Filename location is only populated as the images cycle through.

When images are loaded from subdirectories, the information stored in the Pipeline and Measurements branches become subtly different. Let B be the base directory (either the default image directory or the directory specified as an option to LoadImages). Suppose that N image files are loaded from various subdirectories of B. Let Si be the subdirectory of the i-th file loaded, and let Fi be its file name. Then h.P.Pathname will be the string B; h.P.FilenameIMAGENAME will be a cell array { 'S1/F1', 'S2/F2', ... }; h.M.I.PathName_IMAGENAME will be a cell array { 'B/S1', 'B/S2', ... }; and h.M.I.FileName_IMAGENAME will be a cell array { 'F1', 'F2', ... }.

Which substructures are deleted prior to an analysis run?
Anything stored in handles.Measurements or handles.Pipeline will be deleted at the beginning of the analysis run, whereas anything stored in handles.Settings, handles.Preferences, and handles.Current will be retained from one analysis to the next. It is important to think about which of these data should be deleted at the end of an analysis run

because of the way MATLAB saves variables: For example, a user might process 12 image sets of nuclei which results in a set of 12 measurements ("TotalStainedArea") stored in handles.Measurements.Image. In addition, a processed image of nuclei from the last image set is left in handles.Pipeline.SegmentedNuclei. Now, if the user uses a different module which happens to have the same measurement output name "TotalStainedArea" to analyze 4 image sets, the 4 measurements will overwrite the first 4 measurements of the previous analysis, but the remaining 8 measurements will still be present. So, the user will end up with 12 measurements from the 4 sets. Another potential problem is that if, in the second analysis run, the user runs only a module which depends on the output "SegmentedNuclei" but does not run a module that produces an image by that name, the module will run just fine: it will just repeatedly use the processed image of nuclei leftover from the last image set, which was left in handles.Pipeline.

How do I save the handles structure in a GUI module?
Any changes you make to the handles structure are not kept from one module to the next unless they are saved to the GUI first. This is done in MATLAB by using the command guidata(gcbo,handles), where gcbo is a function which identifies the CellProfiler window to the module. Since the guidata command can only store one variable at a time, be sure to use it on the handles structure only.

*** IMAGE ANALYSIS ***

If you plan to use the same function in two different m-files (e.g. a module and a data tool, or two modules), it is helpful to write a CPsubfunction called by both m-files so that you have only one subfunction's code to maintain if any changes are necessary.

Images loaded into CellProfiler are in the 0 to 1 range for consistency across modules. When retrieving images into your module, you can check the images for proper range, size, color/gray, etc using the CPretrieveimage subfunction.

We have used many MATLAB functions from the image processing toolbox. Currently, CellProfiler does not require any other toolboxes for processing.

The 'drawnow' function allows figure windows to be updated and buttons to be pushed (like the pause, cancel, help, and view buttons). The 'drawnow' function is sprinkled throughout the code so there are plenty of breaks where the figure windows/buttons can be interacted with. This does theoretically slow the computation somewhat, so it might be reasonable to remove most of these lines when running jobs on a cluster where speed is important.

*** ERROR HANDLING ***

* In data tools & image tools:

```
     CPerrordlg(['Image processing was canceled in the ',ModuleName,'
            module because your entry ',ValueX,' was invalid.'])
     return
```

```
* In modules and CPsubfunctions (no need for "return"):
      error('Your error message here.')

* Note:
Always try to make the subfunctions as less likely to have errors as
possible. Whenever you can, have error checks in the calling function
before the subfunction gets called. Since CPsubfunctions use
error('message'), you should try to nest any calls to them in a
try/catch. Plus, this allows you to add more specific information to the
error message (such as where in the calling function did the error
occur). To do this, you can just throw an error whose message has your
additional information together with lasterr (which retrieves the last
error message). In data tools and image tools CPerrordlg('message') and
return is needed because they are usually called independently, and using
error('message') would just stop execution, but would not prompt the user
with the corresponding error message.

*** DISPLAYING RESULTS ***

Each module checks whether its figure is open before calculating images
that are for display only. This is done by examining all the figure
handles for one whose handle is equal to the assigned figure number for
this algorithm. If the figure is not open, everything between the "if"
and "end" is ignored (to speed execution), so do not do any important
calculations there. Otherwise an error message will be produced if the
user has closed the window but you have attempted to access data that was
supposed to be produced by this part of the code.  This is especially
problematic when running on a cluster of computers with no displays. If
you plan to save images which are normally produced for display only, the
corresponding lines should be moved outside this if statement. Also, any
additional uicontrols (popupmenus, pushbuttons) should be designed
using the unit of pixels, since this is standard across platforms unlike
other units such as inches and points.

STEP 1: Find the appropriate figure window. If it is closed, usually none
of the remaining steps are performed.
  ThisModuleFigureNumber = handles.Current.(['FigureNumberForModule',CurrentModule]);
  if any(findobj == ThisModuleFigureNumber)

STEP 2: Activate the appropriate figure window so subsequent steps are
performed inside this window:
  CPfigure(handles,'Image',ThisModuleFigureNumber);
For figures that contain any images, choose 'Image', otherwise choose
'Text'. 'Image' figures will have the RGB checkboxes which allow
displaying individual channels, the InteractiveZoom and CellProfiler
Image Tools menu items, and the Raw/Stretched intensity scale pulldown.

Note: unfortunately there is no convenient way right now to have more
than one figure window per module. We work around this in the case of
IdPrimAutomatic when run in "test mode", for example, by creating a new
window with a special 'Tag' property that allows you to find it again in
subsequent cycles. Having the 'Name' property of the figure window
containing "cycle #" at the end allows CellProfiler to recognize it and
```



```
list it in the Windows menu.
Also note: In general, you should not change figure properties like this:
CPfigure('Tag', 'My figure name')
...because it messes up the menus in the figure window. Use this instead:
set(FigureHandle,'Tag','My figure name');

STEP 3: (only during starting image cycle) Make the figure the proper
size:
  if handles.Current.SetBeingAnalyzed == handles.Current.StartingImageSet
    CPresizefigure('','NarrowText',ThisModuleFigureNumber)
  end
The figure is adjusted to fit the aspect ratio of the images, depending
on how many rows and columns of images should be displayed. The choices
are: OneByOne, TwoByOne, TwoByTwo, NarrowText. If a figure display is
unnecessary for the module, skip STEP 2 and here use:
  if handles.Current.SetBeingAnalyzed == handles.Current.StartingImageSet
    close(ThisModuleFigureNumber)
  end
or simply use the subfunction:
  CPclosefigure(handles,CurrentModule)

Note that in the above we do not use this:
  if handles.Current.SetBeingAnalyzed == 1
... because if the user has chosen the Restart module to resume analysis,
the first image set being processed will not be #1, and yet we want the
figure window to be sized properly.

STEP 4: Display your image:
  ImageHandle = CPimagesc(Image,handles);
This CPimagesc displays the image and also embeds an image tool bar which
will appear when you click on the displayed image. The handles are passed
in so the user's preferences for font size and colormap are used.

*** DEBUGGING HINTS ***

* Use breakpoints in MATLAB to stop your code at certain points and
examine the intermediate results.

* To temporarily show an image during debugging, add lines like this to
your code, or type them at the command line of MATLAB:
      CPfigure
      CPimagesc(BlurredImage, [])
      title('BlurredImage')

* To temporarily save an intermediate image during debugging, try this:
      imwrite(BlurredImage, 'FileName.tif', 'FileFormat');
Note that you may have to alter the format of the image before
saving. If the image is not saved correctly, for example, try
adding the uint8 command:
      imwrite(uint8(BlurredImage), 'FileName.tif', 'FileFormat');

* To routinely save images produced by this module, see the help in
the SaveImages module.
```

* If you want to save images that are produced by other modules but that are not given an official name in the settings boxes for that module, alter the code for the module to save those images to the handles structure and then use the Save Images module.
The code should look like this:
fieldname = ['SomeDescription(optional)',ImgOrObjNameFromSettingsBox];
handles.Pipeline.(fieldname) = ImageProducedBytheModule;
Example 1:
fieldname = ['Segmented', ObjectName];
handles.Pipeline.(fieldname) = SegmentedObjectImage;
Example 2:
fieldname = CroppedImageName;
handles.Pipeline.(fieldname) = CroppedImage;

For General help files:
We have one line of actual code in these files so that the help is visible. We are not using CPhelpdlg because using helpdlg instead allows the help to be accessed from the command line of MATLAB. The one line of code in each help file (helpdlg) is never run from inside CP anyway.

*** RUNNING CELLPROFILER WITHOUT THE GRAPHICAL USER INTERFACE ***

In order to run CellProfiler modules without the GUI you must have the following variables:

handles.Settings.ModuleNames (for all modules in pipeline)
handles.Settings.VariableValues (for all modules in pipeline)
handles.Current.CurrentModuleNumber (must be consistent with pipeline)
handles.Current.SetBeingAnalyzed (must be consistent with pipeline)
handles.Current.FigureNumberForModuleXX (for all modules in pipeline)
handles.Current.NumberOfImageSets (set by LoadImages, so if it is run first, you do not need to set it)
handles.Current.DefaultOutputDirectory
handles.Current.DefaultImageDirectory
handles.Current.NumberOfModules
handles.Preferences.IntensityColorMap (only used for display purposes)
handles.Preferences.LabelColorMap (only used for display purposes)
handles.Preferences.FontSize (only used for display purposes)

You will also need to have the CPsubfunctions folder, since our Modules call CP subfunctions for many tasks. The CurrentModuleNumber needs to be set correctly for each module in the pipeline since this is how the variable values are called. In order to see what all of these variables look like, run a sample analysis and then go to File -> Tech Diagnosis. This will let you manipulate the handles variable in MATLAB.

CellProfiler Help: FastMode

Fast mode can be set in File > Set preferences.

If you uncheck the box you will run in diagnostic mode, where all the intermediate images and calculations for the most recent image cycle are saved in the output file, which drastically increases the output file size. Check the box if you would instead like to run in normal (fast) mode, producing smaller output files.

See also the SpeedUpCellProfiler module.