Using OpenCV with Eclipse (plugin CDT)

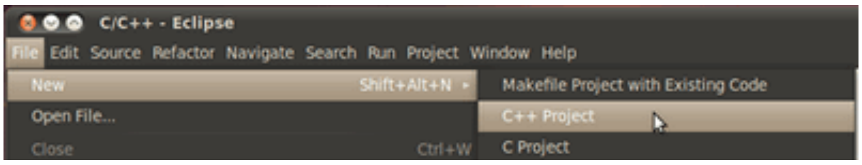
Note: Two ways, one by forming a project directly, and another by CMake

Prerequisites

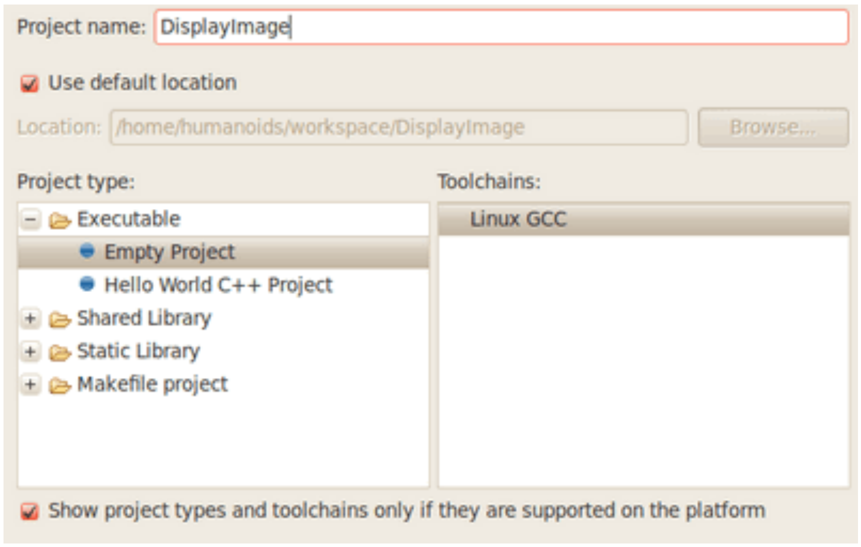
1. Having installed Eclipse in your workstation (only the CDT plugin for C/C++ is needed). You can follow the following steps:
   - Go to the Eclipse site
   - Download Eclipse IDE for C/C++ Developers . Choose the link according to your workstation.
2. Having installed OpenCV. If not yet, go *here*.

Making a project

1. Start Eclipse. Just run the executable that comes in the folder.
2. Go to **File -> New -> C/C++ Project**


C/C++ - Eclipse
File Edit Source Refactor Navigate Search Run Project Window Help
New Shift+Alt+N
Open File...
Close Ctrl+W
Makefile Project with Existing Code
C++ Project
C Project

3. Choose a name for your project (i.e. DisplayImage). An **Empty Project** should be okay for this example.



Project name: DisplayImage

Use default location

Location: /home/humanoids/workspace/DisplayImage

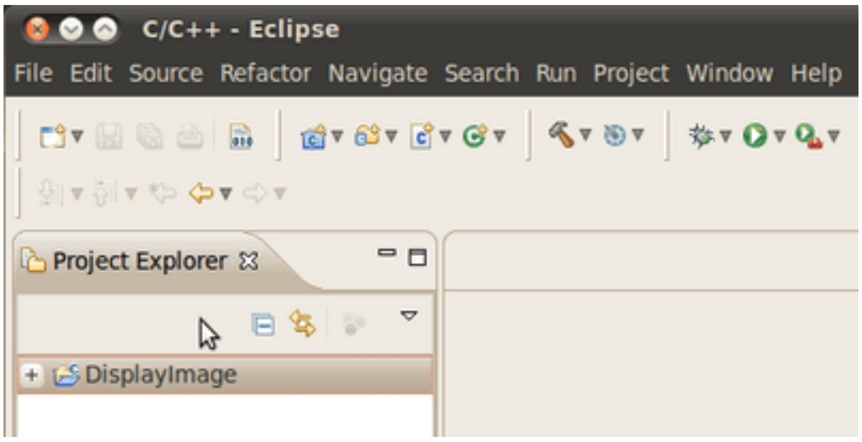
Project type: Executable > Empty Project

Toolchains: Linux GCC

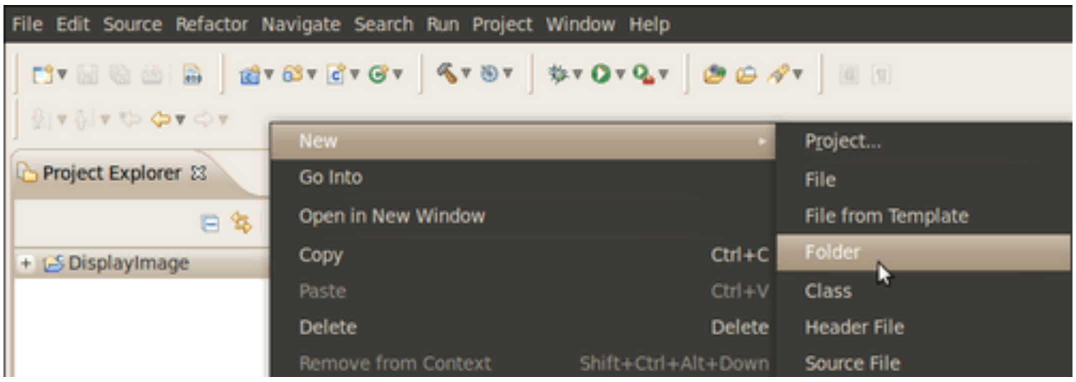
Show project types and toolchains only if they are supported on the platform

4. Leave everything else by default. Press **Finish**.

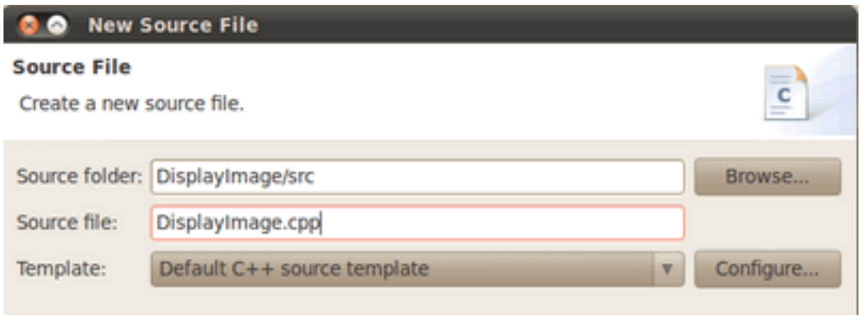
5. Your project (in this case DisplayImage) should appear in the **Project Navigator** (usually at the left side of your window).


C/C++ - Eclipse
File Edit Source Refactor Navigate Search Run Project Window Help
Project Explorer
DisplayImage

6. Now, let's add a source file using OpenCV:
   - Right click on **DisplayImage** (in the Navigator). **New -> Folder** .



   - Name your folder **src** and then hit **Finish**
   - Right click on your newly created **src** folder. Choose **New source file**:
   - Call it **DisplayImage.cpp**. Hit **Finish**


New Source File
Source File
Create a new source file.
Source folder: DisplayImage/src
Browse...
Source file: DisplayImage.cpp
Template: Default C++ source template
Configure...

7. So, now you have a project with a empty .cpp file. Let's fill it with some sample code (in other words, copy and paste the snippet below):

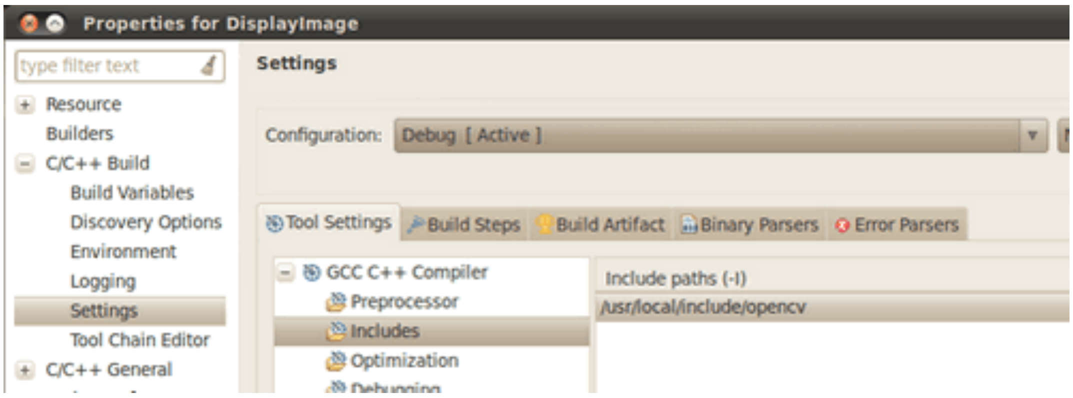
```
#include <cv.h>
#include <highgui.h>
```

```
using namespace cv;

int main( int argc, char** argv )
{
  Mat image;
  image = imread( argv[1], 1 );

  if( argc != 2 || !image.data )
    {
      printf( "No image data \n" );
      return -1;
    }

  namedWindow( "Display Image", CV_WINDOW_AUTOSIZE );
  imshow( "Display Image", image );

  waitKey(0);

  return 0;
}
```

8. We are only missing one final step: To tell OpenCV where the OpenCV headers and libraries are. For this, do the following:

   - Go to **Project–>Properties**
   - In **C/C++ Build**, click on **Settings**. At the right, choose the **Tool Settings** Tab. Here we will enter the headers and libraries info:

     a. In **GCC C++ Compiler**, go to **Includes**. In **Include paths(-I)** you should include the path of the folder where opencv was installed. In our example, this is `/usr/local/include/opencv`.



Configuration: Debug [Active]

Include paths (-I)
/usr/local/include/opencv

Note: If you do not know where your opencv files are, open the **Terminal** and type:

```
pkg-config --cflags opencv
```

For instance, that command gave me this output:

```
-I/usr/local/include/opencv -I/usr/local/include
```

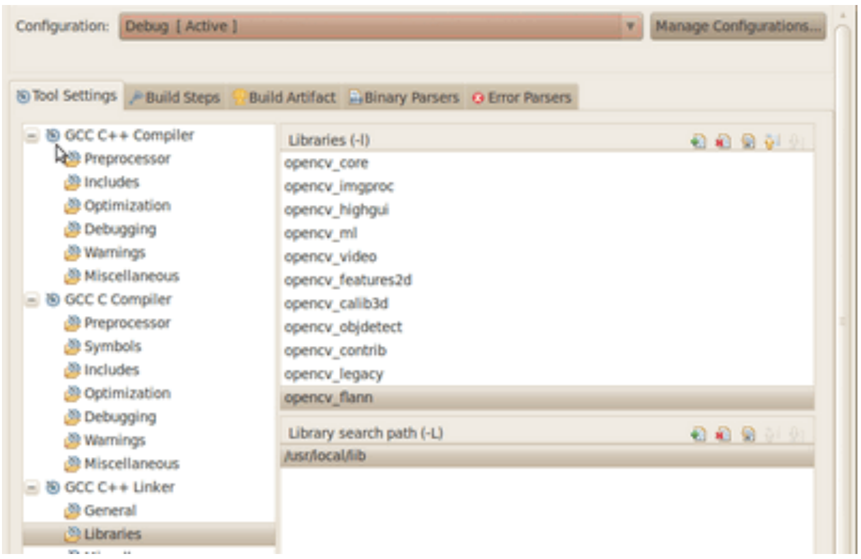
b. Now go to **GCC C++ Linker**,there you have to fill two spaces:

First in **Library search path (-L)** you have to write the path to where the opencv libraries reside, in my case the path is:

```
/usr/local/lib
```

Then in **Libraries(-l)** add the OpenCV libraries that you may need. Usually just the 3 first on the list below are enough (for simple applications) . In my case, I am putting all of them since I plan to use the whole bunch:

opencv_core opencv_imgproc opencv_highgui opencv_ml opencv_video opencv_features2d opencv_calib3d opencv_objdetect opencv_contrib opencv_legacy opencv_flann



Libraries (-l)
opencv_core
opencv_imgproc
opencv_highgui
opencv_ml
opencv_video
opencv_features2d
opencv_calib3d
opencv_objdetect
opencv_contrib
opencv_legacy
opencv_flann

Library search path (-L)
/usr/local/lib

If you don't know where your libraries are (or you are just psychotic and want to make sure the path is fine), type in **Terminal**:

```
pkg-config --libs opencv
```

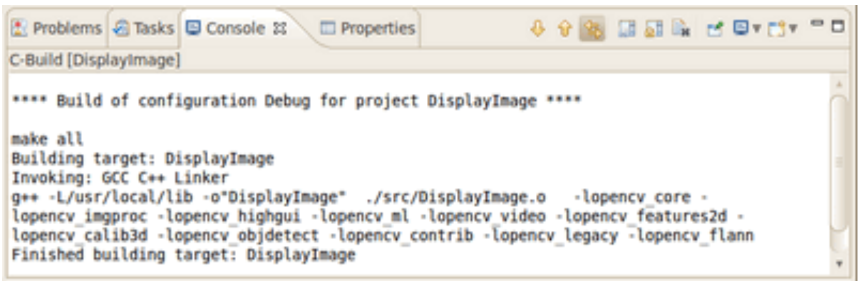
My output (in case you want to check) was: .. code-block:: bash

> -L/usr/local/lib -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_ml -lopencv_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_flann

Now you are done. Click **OK**

- Your project should be ready to be built. For this, go to **Project->Build all**

  In the Console you should get something like



```
**** Build of configuration Debug for project DisplayImage ****

make all
Building target: DisplayImage
Invoking: GCC C++ Linker
g++ -L/usr/local/lib -o"DisplayImage"  ./src/DisplayImage.o   -lopencv_core -
lopencv_imgproc -lopencv_highgui -lopencv_ml -lopencv_video -lopencv_features2d -
lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_flann
Finished building target: DisplayImage
```

  If you check in your folder, there should be an executable there.

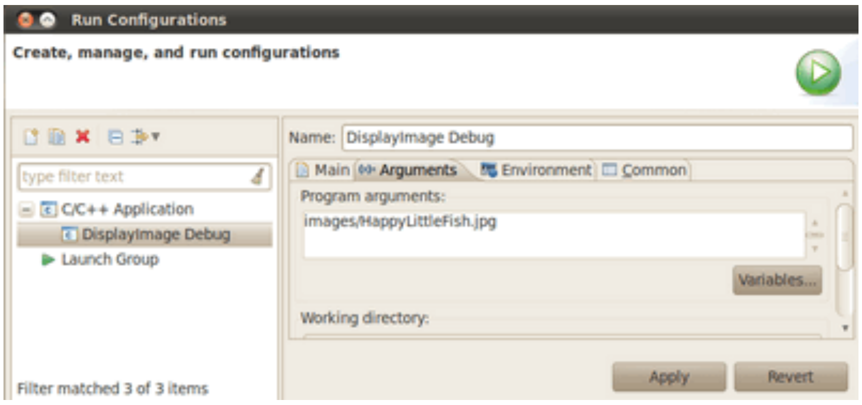
Running the executable

So, now we have an executable ready to run. If we were to use the Terminal, we would probably do something like:

```
cd <DisplayImage_directory>
cd src
./DisplayImage ../images/HappyLittleFish.png
```

Assuming that the image to use as the argument would be located in <DisplayImage_directory>/images/HappyLittleFish.png. We can still do this, but let's do it from Eclipse:

1. Go to **Run->Run Configurations**
2. Under C/C++ Application you will see the name of your executable + Debug (if not, click over C/C++ Application a couple of times). Select the name (in this case **DisplayImage Debug**).
3. Now, in the right side of the window, choose the **Arguments** Tab. Write the path of the image file we want to open (path relative to the workspace/DisplayImage folder). Let's use **HappyLittleFish.png**:

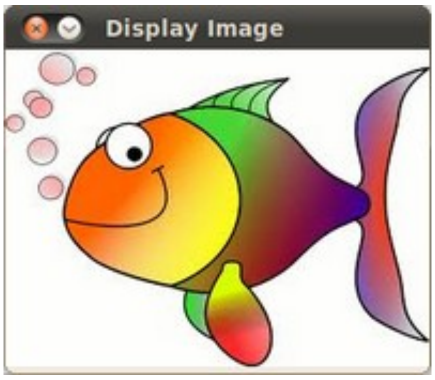


Name: DisplayImage Debug

Program arguments:
images/HappyLittleFish.jpg

Working directory:

4. Click on the **Apply** button and then in Run. An OpenCV window should pop up with the fish image (or whatever you used).



5. Congratulations! You are ready to have fun with OpenCV using Eclipse.

V2: Using CMake+OpenCV with Eclipse (plugin CDT)

Say you have or create a new file, *helloworld.cpp* in a directory called *foo*:

```
#include <cv.h>
#include <highgui.h>
int main ( int argc, char **argv )
{
  cvNamedWindow( "My Window", 1 );
  IplImage *img = cvCreateImage( cvSize( 640, 480 ), IPL_DEPTH_8U, 1 );
  CvFont font;
  double hScale = 1.0;
  double vScale = 1.0;
  int lineWidth = 1;
  cvInitFont( &font, CV_FONT_HERSHEY_SIMPLEX | CV_FONT_ITALIC,
              hScale, vScale, 0, lineWidth );
  cvPutText( img, "Hello World!", cvPoint( 200, 400 ), &font,
             cvScalar( 255, 255, 0 ) );
  cvShowImage( "My Window", img );
  cvWaitKey();
  return 0;
}
```

1. Create a build directory, say, under *foo*: `mkdir /build`. Then `cd build`.

2. Put a *CmakeLists.txt* file in build:

```
PROJECT( helloworld_proj )
FIND_PACKAGE( OpenCV REQUIRED )
ADD_EXECUTABLE( helloworld helloworld.cxx )
TARGET_LINK_LIBRARIES( helloworld ${OpenCV_LIBS} )
```

1. Run: `cmake-gui ..` and make sure you fill in where opencv was built.
2. Then click `configure` and then `generate`. If it's OK, **quit cmake-gui**
3. Run `make -j4` *(the ``-j4`` is optional, it just tells the compiler to build in 4 threads)*. Make sure it builds.
4. Start `eclipse` . Put the workspace in some directory but **not** in `foo` or `foo\\build`
5. Right click in the `Project Explorer` section. Select `Import` And then open the `C/C++` filter. Choose *Existing Code* as a Makefile Project``
6. Name your project, say *helloworld*. Browse to the Existing Code location `foo\\build` (where you ran your cmake-gui from). Select *Linux GCC* in the *"Toolchain for Indexer Settings"* and press *Finish*.
7. Right click in the `Project Explorer` section. Select `Properties`. Under `C/C++ Build`, set the *build directory:* from something like `${workspace_loc:/helloworld}` to `${workspace_loc:/helloworld}/build` since that's where you are building to.
   a. You can also optionally modify the `Build command:` from `make` to something like `make VERBOSE=1 -j4` which tells the compiler to produce detailed symbol files for debugging and also to compile in 4 parallel threads.

1. Done!

Help and Feedback

You did not find what you were looking for?

Ask a question on the **Q&A forum**.
If you think something is missing or wrong in the documentation, please file a **bug report**.