



Main Page

Related Pages

Namespaces ▾

Classes ▾

Files ▾

Examples

Java documentation

Q ▾ Search

OpenCV-Python Tutorials

OpenCV-Python Bindings

How OpenCV-Python Bindings Works?

Goal

Learn:

- How OpenCV-Python bindings are generated?
- How to extend new OpenCV modules to Python?

How OpenCV-Python bindings are generated?

In OpenCV, all algorithms are implemented in C++. But these algorithms can be used from different languages like Python, Java etc. This is made possible by the bindings generators. These generators create a bridge between C++ and Python which enables users to call C++ functions from Python. To get a complete picture of what is happening in background, a good knowledge of Python/C API is required. A simple example on extending C++ functions to Python can be found in official Python documentation[1]. So extending all functions in OpenCV to Python by writing their wrapper functions manually is a time-consuming task. So OpenCV does it in a more intelligent way. OpenCV generates these wrapper functions automatically from the C++ headers using some Python scripts which are located in `modules/python/src2`. We will look into what they do.

First, `modules/python/CMakeFiles.txt` is a CMake script which checks the modules to be extended to Python. It will automatically check all the modules to be extended and grab their header files. These header files contain list of all classes, functions, constants etc. for that particular modules.

Second, these header files are passed to a Python script, `modules/python/src2/gen2.py`. This is the Python bindings generator script. It calls another Python script `modules/python/src2/hdr_parser.py`. This is the header parser script. This header parser splits the complete header file into small Python lists. So these lists contain all details about a particular function, class etc. For example, a function will be parsed to get a list containing function name, return type, input arguments, argument types etc. Final list contains details of all the functions, enums, structs, classes etc. in that header file.

But header parser doesn't parse all the functions/classes in the header file. The developer has to specify which functions should be exported to Python. For that, there are certain macros added to the beginning of these declarations which enables the header parser to identify functions to be parsed. These macros are added by the developer who programs the particular function. In short, the developer decides which functions should be extended to Python and which are not. Details of those macros will be given in next session.

So header parser returns a final big list of parsed functions. Our generator script (`gen2.py`) will create wrapper functions for all the functions/classes/enums/structs parsed by header parser (You can find these header files during compilation in the `build/modules/python/` folder as `pyopencv_generated_*.h` files). But there may be some basic OpenCV datatypes like `Mat`, `Vec4i`, `Size`. They need to be extended manually. For example, a