

---

# **retdec-python Documentation**

*Release 0.6-dev*

**Petr Zemek and contributors**

**Oct 11, 2018**



---

# Contents

---

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Quickstart . . . . .	3
1.2	Library . . . . .	4
1.3	Scripts . . . . .	8
1.4	Contributing . . . . .	11
1.5	Status . . . . .	11
<b>2</b>	<b>Indices</b>	<b>15</b>



retdec-python is a Python library and tools providing easy access to the [retdec.com](https://retdec.com) decompilation service through their public [REST API](#).



## 1.1 Quickstart

This page gives an overview of the library and tools to get you started.

### 1.1.1 Requirements

- Python  $\geq$  3.4 (CPython or PyPy)
- `requests` module for making HTTPS calls to the [retdec.com](https://retdec.com) API

### 1.1.2 Installation

The recommended way of installing is from [Python Package Index \(PyPI\)](https://pypi.org/) with `pip`:

```
$ pip install retdec-python
```

This will install the latest stable version, including all dependencies. You can also install the latest development version directly from [GitHub](https://github.com):

```
$ pip install git+https://github.com/s3rvac/retdec-python
```

### 1.1.3 Prerequisites

To be able to actually use the library and scripts, you need to [register](https://retdec.com) at [retdec.com](https://retdec.com). After that, [log in](#) and click on [Account](#). There, you will find your API key, which is used for authentication.

**Attention:** Be careful not to disclose your API key to anyone! You have to keep it a secret.

## 1.1.4 Library vs Scripts

retdec-python provides both a Python library and scripts. You can either incorporate the library in your own scripts:

```
from retdec.decompiler import Decompiler

decompiler = Decompiler(api_key='YOUR-API-KEY')
decompilation = decompiler.start_decompilation(input_file='file.exe')
decompilation.wait_until_finished()
decompilation.save_hll_code()
```

or you can use the provided scripts for stand-alone file analyses or decompilations:

```
$ decompiler -k YOUR-API-KEY file.exe
v23bmYb67R
-----
Waiting for resources (0%)... [OK]
Pre-Processing:
  Obtaining file information (5%)... [OK]
  Unpacking (10%)... [OK]
Front-End:
  Initializing (20%)... [OK]
[..]
Done (100%)...

Downloading:
- file.c
```

Either way, file.c then contains the decompiled C code:

```
$ cat file.c
//
// This file was generated by the Retargetable Decompiler
// Website: https://retdec.com
// Copyright (c) 2016 Retargetable Decompiler <info@retdec.com>
//
#include <stdio.h>
[..]
```

The library provides support for the [decompilation](#), [fileinfo](#), and [test](#) services. For a more detailed list, see the [status](#) page.

Next, we describe the library in a greater detail. If you wish to learn more about the provided scripts, continue [here](#) instead.

## 1.2 Library

This page describes the retdec-python library and its API.

### 1.2.1 Organization

The base package is retdec. Everything that the library provides is inside this package.



## 1.2.2 Authentication

The library needs to authenticate you to [retdec.com](https://retdec.com). To specify your API key, either pass it as a parameter when creating a resource:

```
decompiler = retdec.decompiler.Decompiler(api_key='YOUR-API-KEY')
```

or set the `RETDEC_API_KEY` environment variable:

```
$ export RETDEC_API_KEY=YOUR-API-KEY
```

An advantage of the environment variable is that you do not need to specify the API key every time you use the library:

```
decompiler = retdec.decompiler.Decompiler()
```

## 1.2.3 Error Handling

The library uses exceptions to signalize errors. The base class is `retdec.exceptions.RetdecError`, which you can use to catch all custom exceptions raised by the library:

```
try:
    # ...
except retdec.exceptions.RetdecError as ex:
    # Handle the error.
```

You can also catch specific exceptions, e.g. `retdec.exceptions.AuthenticationError`, and react on them. See the `retdec.exceptions` module for a list of all custom exceptions.

## 1.2.4 Decompiler

The `retdec.decompiler` module provides access to the [decompilation service](#). It allows you to decompile binary files into a high-level language representation, such as C.

### Creating a Decompiler

The decompiler is represented by the `retdec.decompiler.Decompiler` class:

```
decompiler = retdec.decompiler.Decompiler(api_key='YOUR-API-KEY')
```

### Starting a Decompilation

To start a decompilation of a file, call `start_decompilation()` on the created decompiler:

```
decompilation = decompiler.start_decompilation(input_file=FILE)
```

`FILE` is either a path to the file or a file-like object. For a complete list of parameters that you can use when starting a decompilation, see the description of `start_decompilation()`.

The returned object is an instance of `retdec.decompilation.Decompilation`.

## Waiting For the Decompileation To Finish

After the `start_decompilation()` call above returns, the decompilation has been automatically started. To wait until it finishes, call `wait_until_finished()`:

```
decompilation.wait_until_finished()
```

If you want to track the decompilation progress (e.g. by showing a progress bar or displaying the log), you can pass a callback function to `wait_until_finished()`:

```
def show_progress(decompilation):
    print(decompilation.get_completion())

decompilation.wait_until_finished(
    callback=show_progress
)
```

When the status of the decompilation changes (e.g. it moves to another phase), the callback is automatically called with the decompilation being passed as its parameter.

## Downloading Outputs

To obtain the generated high-level language (HLL) code as a string, call `get_hll_code()`:

```
print(decompilation.get_hll_code())
```

Alternatively, you can call `save_hll_code()`, which obtains and saves the generated HLL code into the given directory:

```
decompilation.save_hll_code('/home/user/downloads')
```

Apart from obtaining the HLL code, you can also get the disassembled code, control-flow graphs, call graph, archive with all the outputs or, in the `c` mode, the compiled version of the input C file. See the description of `Decompilation` for more details.

For a complete example, take a look the `retdec/tools/decompiler.py` file. It is an implementation of the *Decompiler* script.

### 1.2.5 Fileinfo

The `retdec.fileinfo` module provides access to the `file-analyzing service`. It allows you to obtain information about binary files.

#### Creating an Analyzer

The analyzer is represented by the `retdec.fileinfo.Fileinfo` class:

```
fileinfo = retdec.fileinfo.Fileinfo(api_key='YOUR-API-KEY')
```

#### Starting an Analysis

To start an analysis of a file, call `start_analysis()` on the created analyzer with a file to be analyzed:

```
analysis = fileinfo.start_analysis(input_file=FILE)
```

FILE is either a path to the file or a file-like object. Optionally, you can pass the following parameters:

- `verbose=True` – makes the analysis obtain all available information about the file.
- `output_format=json` – causes the output from the analysis to be in the **JSON** format instead of in the plain format.

The returned object is an instance of `retdec.analysis.Analysis`.

### Waiting For the Analysis To Finish

After the `start_analysis()` call above returns, the analysis has been automatically started. To wait until it finishes, call `wait_until_finished()`:

```
analysis.wait_until_finished()
```

### Obtaining the Results of the Analysis

To obtain the output from the analysis, call `get_output()`:

```
print(analysis.get_output())
```

For a complete example, take a look at the `retdec/tools/fileinfo.py` file. It is an implementation of the *Fileinfo* script.

## 1.2.6 Test

Access to the `testing service` is provided by the `retdec.test` module.

### Authentication

To check whether you can authenticate successfully, use `retdec.test.Test.auth()`:

```
test = retdec.test.Test(api_key='YOUR-API-KEY')
try:
    test.auth()
    print('authentication succeeded')
except retdec.exceptions.AuthenticationError as ex:
    print('authentication failed:', ex)
```

### Parameter Passing

To check that parameters are passed correctly when performing requests to the `retdec.com API`, use `retdec.test.Test.echo()`:

```
test = retdec.test.Test(api_key='YOUR-API-KEY')
result = test.echo(param='value')
print(result) # Prints {'param': 'value'}.
```

## 1.3 Scripts

This page describes the `retdec-python` scripts and their usage.

Currently, there are two scripts: `decompiler` and `fileinfo`. They provide access to the [decompilation](#) and [file-analyzing](#) services, respectively.

### 1.3.1 Authentication

The scripts need to authenticate you to [retdec.com](#). To specify your API key, either use the `-k KEY` or `--api-key KEY` parameter:

```
$ decompiler -k YOUR-API-KEY file.exe
```

or set the `RETDEC_API_KEY` environment variable:

```
$ export RETDEC_API_KEY=YOUR-API-KEY
$ decompiler file.exe
```

An advantage of the environment variable is that you do not have to specify the API key every time you run a script.

### 1.3.2 Decompiler

The `decompiler` script provides access to the [decompilation service](#). It allows you to decompile binary files into a high-level language representation, such as C.

#### Usage

```
$ decompiler [OPTIONS] FILE
```

Output files are stored into the same directory where the input file is located. For example, if the input file is `dir/prog.exe`, then the decompiled code in the C language is saved as `dir/prog.c`. You can override the output directory by using the `-o/--output-dir` parameter.

#### Options

See the [official documentation](#) for more details.

- `-a ARCH, --architecture ARCH` – Architecture to force when (de)compiling. Supported architectures: `x86, arm, thumb, mips, pic32, powerpc`.
- `-b, --brief` – Print fewer information during the decompilation.
- `-c COMPILER, --compiler COMPILER` – Compiler to use when compiling input C source files. Supported compilers: `gcc, clang`.
- `-C LEVEL, --compiler-optimizations LEVEL` – Optimization level to use when compiling input C source files. Supported levels: `O0, O1, O2, O3`.
- `--endian` – Endianness of the machine code (`bin` and `raw` modes only). Supported endians: `little, big`.
- `-f FORMAT, --file-format FORMAT` – File format to force when compiling input C source files. Supported formats: `elf, pe`.

- `-g, --compiler-debug` – Compile the input C file with debugging information (i.e. passes the `-g` flag to the used compiler).
- `-s, --compiler-strip` – Strip the compiled C file prior its decompilation.
- `-k KEY, --api-key KEY` – Specifies the API key to be used.
- `-l LANGUAGE, --target-language LANGUAGE` – Target high-level language. Supported languages: `c`, `py`.
- `--graph-format FORMAT` – Format of the generated call and control-flow graphs. Supported formats: `png`, `svg`, `pdf`.
- `-m MODE, --mode MODE` – Decompilation mode. **Supported modes:** `bin`, `c`, and `raw`. By default, the script performs an automatic detection based on the extension of the input file.
- `-o DIR, --output-dir DIR` – Save the outputs into this directory.
- `-p FILE, --pdb-file` – PDB file associated with the input file.
- `-q, --quiet` – Print only errors, nothing else (not even progress).
- `-V, --version` – Print the script and library version.
- `--var-names STYLE` – Naming style for variables. Supported styles: `readable`, `address`, `hungarian`, `simple`, and `unified`.
- `-O LEVEL, --optimizations LEVEL` – Level of optimizations performed by the decompiler. **Supported levels:** `none`, `limited`, `normal`, and `aggressive`.
- `-K, --keep-unreach-funcs` – Decompile all functions, even if they are not reachable.
- `--only-funcs` – Decompile only the given functions (a comma-separated list of function names, e.g. `func1, func2`).
- `--only-ranges '`  – Decompile only the given address ranges (a comma-separated list of address ranges, e.g. `0x100-0x200, 0x500-0x600`).
- `--decoding` – What should be decoded in a selective decompilation? Supported types: `everything`, `only`.
- `--no-addresses` – Disable the emission of addresses in comments in the generated code.
- `--raw-entry-point` – Virtual memory address where execution flow should start in the machine code (`raw mode only`).
- `--raw-section-vma` – Address where the section created from the machine code will be placed in virtual memory (`raw mode only`).
- `--ar-index` – Index of the object file in the input archive to be decompiled when decompiling an archive.
- `--ar-name` – Name of the object file in the input archive to be decompiled when decompiling an archive.
- `--with-cg` – Generate a call graph when the decompilation ends.
- `--with-cfgs` – Generate call graphs for all functions when the decompilation ends.
- `--with-archive` – Generate an archive containing all decompilation outputs when the decompilation ends.

## Example

```
$ decompiler -k YOUR-API-KEY file.exe

v23bmYb67R
-----

Waiting for resources (0%)... [OK]
Pre-Processing:
  Obtaining file information (5%)... [OK]
  Unpacking (10%)... [OK]
Front-End:
  Initializing (20%)... [OK]
[..]
Done (100%)...

Downloading:
- file.c
```

`file.c` then contains the decompiled C code.

### 1.3.3 Fileinfo

The `fileinfo` script provides access to the [file-analyzing service](#). It allows you to obtain information about binary files.

#### Usage

```
$ fileinfo [OPTIONS] FILE
```

#### Options

- `-k KEY, --api-key KEY` – Specifies the API key to be used.
- `-f FORMAT, --output-format FORMAT` – Format in which the output should be printed. Available formats are `plain` (plain text; the default) and `json` (JSON).
- `-v, --verbose` – Print all available information about the file.
- `-V, --version` – Print the script and library version.

#### Example

```
$ fileinfo -k YOUR-API-KEY file.exe

Input file           : file.exe
File format          : PE
File class           : 32-bit
File type            : Executable file
Architecture         : x86 (or later and compatible)
Endianness           : Little endian
Entry point address  : 0x4014e0
Entry point offset   : 0x8e0
Entry point section name : .text
```

(continues on next page)

(continued from previous page)

```
Entry point section index: 0
Bytes on entry point      : 31ed5e89e183e4f05054526860c1040868f0c00408515668
Detected compiler/packer : GCC (x86_64-unknown-linux-gnu) (4.7.2) (100%)
```

## 1.4 Contributing

Any contributions are welcomed! I will be very glad to get your feedback, [pull requests](#), [issues](#), or just a simple *Thanks*. Feel free to contact me for any questions you might have!

### 1.4.1 Coding Style

The code should be [PEP8](#) compliant, except for line length, which may be greater than 79 when suitable (but never exceeding 100 characters).

### 1.4.2 Testing

The code is 100% covered with unit tests. When you make a pull request, please include unit tests for your code to keep the coverage at 100%.

### 1.4.3 Make Targets

- Project documentation can be generated by running `make docs` (you need to have [sphinx](#) and [sphinx\\_rtd\\_theme](#) installed).
- To run unit tests, execute `make tests` (you need to have [nose](#) installed).
- Test coverage can be generated by executing `make tests-coverage` (once again, you need to have [nose](#) installed).
- To ensure that the code complies to [PEP8](#), execute `make lint` (you need to have [flake8](#) installed).

See the contents of [Makefile](#) to for all the possible targets.

## 1.5 Status

A summary of the supported parts of the [retdec.com API](#).

### 1.5.1 Decompiler

The decompilation service.

- [Starting a new decompilation](#)
- [Decompilation modes](#)
  - `bin`
  - `c`
  - `raw`

- Input files
  - input
  - pdb
- Decompilation parameters
  - Mode-independent parameters
    - \* target\_language
    - \* graph\_format
    - \* decomp\_var\_names
    - \* decomp\_optimizations
    - \* decomp\_unreach\_funcs
    - \* decomp\_emit\_addresses
    - \* generate\_cg
    - \* generate\_cfgs
    - \* generate\_archive
  - Parameters for the bin mode
    - \* architecture
    - \* endian
    - \* sel\_decomp\_funcs
    - \* sel\_decomp\_ranges
    - \* sel\_decomp\_decoding
    - \* ar\_index
    - \* ar\_name
  - Parameters for the raw mode
    - \* architecture
    - \* endian
    - \* raw\_entry\_point
    - \* raw\_section\_vma
  - Parameters for the c mode
    - \* architecture
    - \* file\_format
    - \* comp\_compiler
    - \* comp\_optimizations
    - \* comp\_debug
    - \* comp\_strip
- Checking status
  - general (running, finished, etc.)



- completion
- phases
  - \* part
  - \* name
  - \* description
  - \* completion
  - \* warnings
- cg
- cfgs
- archive
- Obtaining outputs
  - hll
  - dsm
  - cg
  - cfgs
  - archive
  - binary
- Error reporting

## 1.5.2 Fileinfo

The file-analyzing service.

- Starting a new analysis
- Optional parameters
  - output\_format
  - verbose
- Checking status
  - general (running, finished, etc.)
- Obtaining output
- Error reporting

## 1.5.3 Test

The testing service.

- Authentication
- Parameter passing



## CHAPTER 2

---

### Indices

---

- genindex
- modindex