**Para-C:** **C-based Coding designed to be simple and fast**

LUNA KLATZER, HTBLA Leonding

LORENZ HOLZBAUER, HTBLA Leonding

Introduction

Para-C (From Greek Origin: Beside C [[1]](#endnote-1)) is a programming language designed to integrate C, compile to C and serve as an extension to C with additional functionality, simplification and helper tools. Including adding more features, like new built-in Macros, simplified or new functions, OOP-structures (Under consideration), more straightforward array, list and malloc-handling, expanded data types and additional project-management features.

The compiler will take the Para-C code to compile it down to simple C with the integrated functionality. That means that programming in Para-C will be similar but simpler and well looking due to the simplifications, new structures, keywords and helper functions. Syntax-wise Para-C will still lay onto C to avoid causing issues with more compiler code that would be required for a new syntax that can’t be easily integrated into the C-syntax. So newer structures won’t look so new, and possibly similar to C# or C++, like data-types, one-liners, overloads and getters etc.

Furthermore, formatting and non-fetal syntax warnings will also be reported, as a help/motivator to avoid causing inconsistent writing and style. Including possibly harder conventions, that will try to improve on the loose C-conventions, which are more open to writing code. That means Para-C will introduce more conventions regarding naming, type declarations, formatting, commenting and will likely also adopt a few Python conventions[[2]](#endnote-2) and integrate some ideas of the Zen of Python[[3]](#endnote-3)(Since the Compiler is also written in Cython).

Content

[1. Base structure 4](#_Toc72530332)

[1.1 Implementation 4](#_Toc72530333)

[1.2 File structure 5](#_Toc72530334)

[1.2.1 Importing PARA-files 6](#_Toc72530335)

[1.2.2 Importing C-libraries or headers 6](#_Toc72530336)

[1.2.3 Project Structure Example 7](#_Toc72530337)

[1.3 The Parac Core library 7](#_Toc72530338)

[1.4 Built-In Identifiers 8](#_Toc72530339)

[1.4.1 Magic Values in the Para-C source code 8](#_Toc72530340)

[1.4.2 Magic Values in the C source code 8](#_Toc72530341)

[1.5 In-Code Exceptions 8](#_Toc72530342)

[2. Compiler and logical Structure of Para-C 8](#_Toc72530343)

[2.1 Structure 8](#_Toc72530344)

[2.2 Compiler Exceptions 8](#_Toc72530345)

[2.2.1 Error-Codes 8](#_Toc72530346)

[3. Para-C Language Reference 9](#_Toc72530347)

[4. Usage-examples of Para-C 9](#_Toc72530348)

[4.1 Project Configuration 9](#_Toc72530349)

[4.1.1 Using the Config file parac-config.json 9](#_Toc72530350)

[4.2 Using C-Code inside Para-C 9](#_Toc72530351)

[4.2.1 Using Standard C-code 9](#_Toc72530352)

[4.2.2 Using C-libraries 9](#_Toc72530353)

[4.2.3 Restrictions of Para-C 9](#_Toc72530354)

[Endnotes 10](#_Toc72530355)

# Base structure

The structure of Para-C will closely lean to the C-Structure (upwards from C11-Standard) but still have its independent system apart from it. That means it will include its own:

* Structure for its Parac-modules and C-modules (*See* [*File Structure*](#_File_structure))
* Import-structure (Bases on C for compatibility)
* Name mangling (Relative to entry-point of the program)
* Exception handling
* Variable System (Required for unique handling and exceptions)
* Built-in macros[[4]](#endnote-4)
* Built-in functions[[5]](#endnote-5) (which will partly replace the C-functions for easier handling)
* OOP-Structure using GObject[[6]](#endnote-6)

## Implementation

The language serves the main purpose of providing new features for the Base-C language, meaning that any structure used will derive from compiled C-code and library-code that provides that functionality. This language library of Para-C is called the Parac Base Library, which is written in C and for higher-level areas Para-C.

The PBL is categorised into three parts:

* Core Library – The code required for the Para-C keywords, functions, identifiers and additional core functionality.
* Built-In Library – Built-in functions that are automatically available in the Para-C code (Imported in the C-code).
* Extension Library – Extension Functions and identifiers for specific use cases. (Implementations can be at their core C, but the overhead will always be written in Para-C, meaning it will be compiled as well at runtime, but only if it was imported)

The base modules and any additionally used structure will be imported into the project at the top of the file. These imports will be separated from the C-imports of the user which are not associated with the Parac Built-In Library.

Any Para-C identifiers in the C-code will have a clear prefix “\_\_pbl\_” and the suffix “\_\_” (pbl = parac base library), and are going to be signalised if needed, using comments to separate user and compiler code.

*Note: To not be confused with the reserved identifiers in C, the Para-C identifiers have the suffix of two underscores instead of just the regular prefix of one or two which are reserved by the C-conventions. (See C11 – 7.1.3 Reserved Identifiers[[7]](#endnote-7))*

If the Para-C compiler declares new variables, calls new functions or updates values that are not part of the user code, but required for other functionality, such as new keywords, special function calls etc. the compiler will attempt to comment with additional information on these lines and separate them clearly from the user code to avoid that code gets mixed up or confused.

## File structure

The file structure of a Para-C project is similar to C and works around the compiler, the PBL, the compiler libraries and the project module. Still due to the structure of Para-C, importing in a project will be different from a regular C/C++ project and depend on the compiler-generated \_\_parac\_\_.h header file, which will define important identifiers inside Para-C which are used throughout the compiled code. That means the compiler will fetch all required imports, paths and additional data and insert them into the file. The file will be placed at the highest level of the project hierarchy, meaning the defined project\_path will be used to insert it.

This header file can be modified if the user intends to do so. Still, it is discouraged to change the \_\_parac\_\_.h file in code, and instead use the universal non-os-dependent parac-config.json file *(See* [*Project Configuration*](#_Using_the_Config)*)*. The file should only be changed if required for specific changes that need to be done that are not available in the normal parac-config.json file. This is because the \_\_parac\_\_.h is configured for the specific system, where the compilation was run. Import paths will likely or almost always not work on other systems and the program will fail to compile or unwanted results are going to occur.

### Importing PARA-files

*(For the sake of clearness, importing and including is used here in the same context and way)*

The standard in-code importing system will not be different from C and will base on a standard header file containing all public available identifiers of the file. This header file can be either included inside your file or another header, which can then be included as well.

Example:

#include “main.ph”

The only difference Para-C is introducing is the way you handle the name mangling and possible duplicate identifier names. That means Para-C will introduce new syntax to handle specific cases and “rename” the identifiers to avoid duplicate naming. Renaming is here in quotation marks since due to the mangling there can never be the case that a variable has the same name as another variable from another file. Still, in the user code, the mangling is not yet applied meaning that importing any header containing a variable declaration with an already existing name, the compiler will be unable to identify which variable is the “correct” one. In this case, you can reorder your code and use a #define to redefine your variable, but for the sake of easier readability it will use the following syntax for renaming imported values:

#from “<header-file>” include <variable-name> as <new-name>

This will “rename” the variable or signalise the compiler that the new variable name will point to that mangled name. The mangled variable will still be imported in the C-version of the code, but in the Para-C version, it will no longer interfere with existing variables.

### Importing C-libraries or headers

Since Para-C is based on C and backwards compatibility, C-code can be easily included into Para-C, by just either importing the library/header which should be available in the standard C-library or including the header file specified. Still, it will require a comment next to import to signalise the compiler its C-code:

* One-Liner: // -\*- lang: C -\*-
* Multi-Line: At the top and bottom of the imports // -\*- lang: C -\*-

The compiler will then also go through the code and possibly change lines of code to fit the functionality of Para-C and not cause unwanted issues. Still, the compiler will not do any major changes and attempt to do as little as possible to keep the functionality alive as wanted. (Since Para-C is not too different from C and C++, C++ code can be included, but due to restrictions of C++, it might not work and break.)

Example:

* Standard C-library module:

#include <stdbool.h> // -\*- lang: C -\*-

* C-header file:

#include “c\_<path\_to\_header>.h” // -\*- lang: C -\*-

* Multi-line Import:

// -\*- lang: C -\*-

#include <stdio.h>

#include <stdbool.h>

// -\*- lang: C -\*-

### Project Structure Example

A simple project structure w

## The Parac Core library

The Para-C core library is, as already explained in the implementation section ([*Implementation of Para-C inside C*](#_Implementation_of_Para-C)), the base for the Para-C programming language. If Para-C functionality is used inside the code that does not exist inside C, the associated core file/library will be imported and used. This means the user does not have to import any headers themselves, since the compiler automatically will import all core library headers that are needed. These imports won’t be inserted into every file, but inserted into the project-wide header file \_\_parac\_\_.h. This header file will be imported into every resulting C file, meaning if a package is used it will be available in every other file. For clarification reasons, the compiler will still log errors for imports for unknown identifiers even if they are imported in another file.

## Built-In Identifiers

Inside Para-C, like in regular C, macros and reserved identifiers are used to provide additional help, store program-vital data and get information about the file, program, function etc. The identifiers are categorised into library functions, magic identifiers and magic constants. (Similar to the Python magic methods and double underscore definition)

### Magic Values in the Para-C source code

### Magic Values in the C source code

## In-Code Exceptions

# Compiler and logical Structure of Para-C

## Structure

## Compiler Exceptions

Exceptions in the Para-C compiler

### Error-Codes

* **1\*\* Internal Errors:**

101 – Interrupt Error: The compiler received an interrupt while running. (derives from the Python BaseException KeyboardInterrupt)

102 – Config Not Found: The parac-config.json file for the project was not found, which is responsible for configuring the Project and compiler.

* **2\*\* File Exceptions:**

200 – File Error: General File Exception in the Compiler

201 – File Permission Error: Failed to access (read, write) to existing file due to missing permissions

202 – File not Found: The File was not found and does not exist! If the file can’t be seen it will be treated as well as a File not Found Error.

203 – Is Directory: File is a directory

* **9\*\* Other Exceptions:**

901 – Unknown Error: Received an unknown exception while running.

# Para-C Language Reference

*(This part of the document serves as the reference for all keywords, identifiers, functions etc. that are added in Para-C and will provide information on how to properly use them)*

# Usage-examples of Para-C

## Project Configuration

### Using the Config file parac-config.json

## Using C-Code inside Para-C

### Using Standard C-code

### Using C-libraries

### Restrictions of Para-C

Endnotes

1. Para meaning and origin: [[here]](https://en.wiktionary.org/wiki/%CF%80%CE%B1%CF%81%CE%AC#Preposition) [↑](#endnote-ref-1)
2. PEP8 – Conventions of Python: [[here]](https://www.python.org/dev/peps/pep-0008/) [↑](#endnote-ref-2)
3. PEP20 – Zen of Python: [[here]](https://www.python.org/dev/peps/pep-0020/) [↑](#endnote-ref-3)
4. List of Pre-defined macros in C: [[here]](https://gcc.gnu.org/onlinedocs/cpp/Predefined-Macros.html) [↑](#endnote-ref-4)
5. List of Built-in C-functions: [[here]](https://www.tutorialspoint.com/ansi_c/c_function_references.htm) [↑](#endnote-ref-5)
6. Introduction to GObject: [[here]](https://www.freedesktop.org/software/gstreamer-sdk/data/docs/latest/gobject/howto-gobject.html) [↑](#endnote-ref-6)
7. C11– Draft April 12, 2011: [[here]](http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf) [↑](#endnote-ref-7)