**INTELLIGENT CYBER-PHYSICAL SYSTEM**

**DOCUMENTATION**

# OVERVIEW

This documentation serves as a documentation for the Intelligent Cyber Physical System (iCPS). The purpose of this document is such that it allows a better understanding of any one new to the development of the iCPS and sets a standard and baseline for those who are currently working on this project. This document is definitely not exhaustive and is open to changes as the development of the system progresses.

# INTRODUCTION

This program, Intelligent Cyber-Physical System, was started in August 2018. The goal of this program is to design a system that consolidates interoperable modular components to develop, execute and analyze human-centered CPS(s).

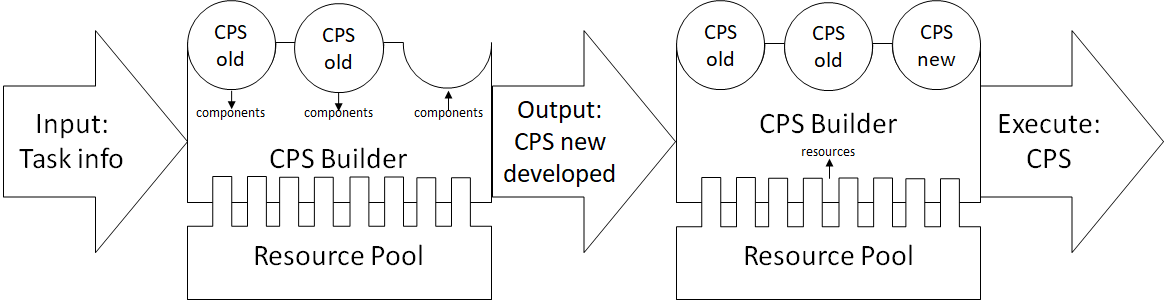


Figure 1 Overview of iCPS.

CPS consists of independent computational components and physical components where intrinsic interaction happens between them as one system Cyber communication may happen either in wired or wireless manner. In this program, the CPS components are classified in this manner: process, physical resource, cyber resource and location.

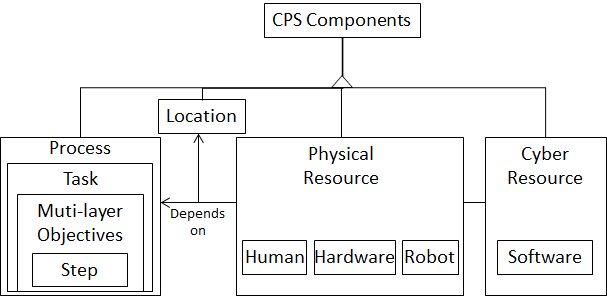


Figure 2 CPS components.

# CREATION OF THE iCPS

## Framework

Flask

The framework used to create the iCPS is Flask, a lightweight web application framework designed to make web creation getting started quick and easy, with the ability to scale up to complex applications. It is based on the projects Werkzeug and Jinja2, both of these projects use Python as the programming language. Flask is considered as a micro framework which does not require particular tools or libraries. It has no database abstraction layer, form validation and other pre-existing common functions. However, Flask supports extensions that can add application features to support and increase the features of the created application. These extensions include object-relational mapping, form validation, upload handling, authentication technology and this list is not exhaustive.

More information on the documentation of the Flask framework can be found at:

<https://www.palletsprojects.com/p/flask/>

## Database

MongoDB

For the database, MongoDB is used which is a cross-platform document-orientated database program. The main features of MongoDB include: Ad hoc queries, indexing, replication, load balancing, file storage, aggregation, server-side JavaScript execution, capped collections, and transaction.

More information on the documentation of MongoDB and its functions can be found at:

[**https://www.mongodb.com/**](https://www.mongodb.com/)

## File Structure

Due to the framework of Flask, the files are called in a particular fashion and the file structure for the iCPS is as shown.

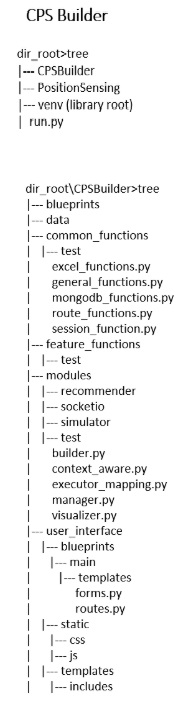


Figure 3 File structure of iCPS.

In general, the templates are to be stored in a folder called templates and the static files are stored in a folder called static. The templates can be further grouped up with the help of a feature by Flask called Blueprint and the iCPS is grouped by its main features mainly: Users, Main, Start, Processes, Resources and History. And the functions and modules to handle the data are grouped according to their uses and area of usage.

# CODING STANDARDS

In total, 4 different coding languages are used for the creation of the iCPS. Python for the functions, modules, routes and forms; JavaScript and CSS for the static files and HTML5 for the templates. In this section, the coding standards are set to allow better readability and standardization of the codes to allow not only the creator of the file to understand but to also allow others to quickly understand the code in the event of troubleshooting or assisting in the development of the system. Pycharm is the preferred IDE for this application.

## GENERAL

Line Properties

Due to the file structure, the files are not restricted to the number of lines for each individual file but is best coded with minimal lines to ease the troubleshooting process and making it understandable

Quotation Marks

Double quotation marks (“” ) are used throughout the project. In the event, there is a quotation mark inside a string, the single quotation mark is used to avoid backlashes in the string. The single quotation mark is used on outside of the string.

Function Arrangements

Functions that are called in another script/file are arranged on top, followed by functions that are only called within the script/file.

## PYTHON

Python governs the utils, threads, modules and routes. In this section, the coding standards are split into their respective segments. The python coding standard is adapted from PEP 8 style guide. (<https://www.python.org/dev/peps/pep-0008/>)

### Common

Coding Style

*Comments:*

**Script Description.** At the beginning of the script, add docstrings to describe the project title, the functions in a single line and the author’s name. Leave one line spacing after the block comment.

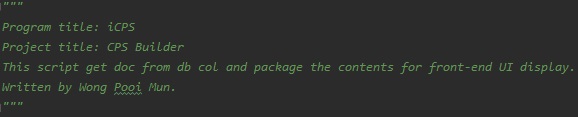


Figure Example of script description for Python.

**Class Description.** At the beginning of each class, add a one-line indented docstrings to describe the functions of the class. Start the comment with a verb of a plural object and end it with a full stop.

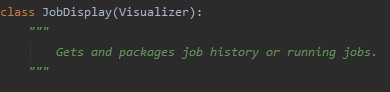


Figure Example of class description for Python.

Fig 5. Example of Class Description for Python

**Function Description.** At the beginning of each function, add docstrings to describe the function. Start the descriptions with a single line summary. Start the summary with a verb of a singular object and end it with a full stop. If detailed descriptions of the function are to be added, leave a blank line, then add each sentence starting on a new line. If parameters and results are to be described, leave a blank line before the description.

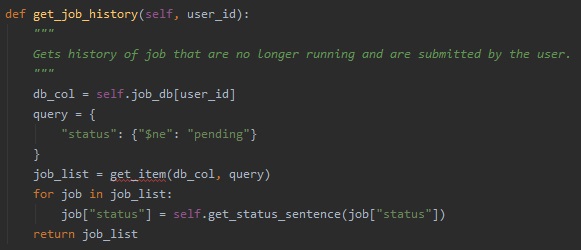


Figure Example of function description for Python.

**Line description.** Add inline comments at the end of code lines that require further explanation. Before the comment leave four blank spaces, followed by a “#”, and then the comments. Do not add inline comments if the code line state the obvious.

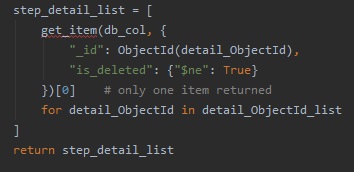


Figure Example of line description for Python.

### Utils

Nomenclature

Files should be named after the functionality of the functions in the script. Naming should be in singular, in lower case and start with a verb. Underscore (\_) should be used between each word of the name. For example, get\_sentence.

Coding Style

*Column placement****:***

Lines are kept within 120 characters and characters after that should be wrapped accordingly

*Indentation****:***

Indentations, wherever appropriate are kept 4 characters. The tab key can also be used to create the indentations

*Import*

from <python package> import <property>

import <python package> as <var>

import <python package>

One blank line spacing before importing python packages.

*Function:*

Naming is in lowercase, use underscore ( \_ ) if there are more than one word. Avoid unnecessary verbs (e.g. is, are). Aim towards readability.

Two blank lines spacing before the functions.

Two blank lines between each function.

*End of file:*

One blank line at the end of the file

### Modules

Nomenclature

Files should be named after the parent class of the script. Naming should be in singular and a noun. The first alphabet of each word should be in upper case, whereas the rest in lower case. There should be no spacing between each word. For example: ProcessManager.

Coding Style

*Column placement****:***

Lines are kept within 120 characters and characters after that should be wrapped accordingly

*Indentation****:***

Indentations, wherever appropriate are kept 4 characters. The tab key can also be used to create the indentations

*Import:*

In modules, only import files from functions folder. Files from other folders should only be imported when necessary, and there is no way around it. Import should follow alphabetical order.

from <file location in functions folder> import \*

from run import <object>

import config

from <python package> import <property>

import <python package> as <var>

import <python package>

import logging

One blank line spacing before importing files.

One blank line spacing before importing python packages.

One blank line spacing before importing logging, followed by initializing the logger.

*Function:*

Naming is in lowercase, use underscore ( \_ ) if there are more than one word. Avoid unnecessary verbs (e.g. is, are). Aim towards readability.

*Classes:*

Naming is in lowercase, use underscore ( \_ ) if there are more than one word. Avoid unnecessary verbs (e.g. is, are). Aim towards readability.

One blank two lines spacing before the class.

One blank line spacing before the functions in the classes.

*Initialization:*

Inputs should include client of MongoDB, test=False and demo=False. Other inputs can be included if necessary.

*End of file:*

One blank line at the end of the file

### Threads

Nomenclature

Files should be named after the content that the thread processes. Naming should be in singular, in lower case and start with a verb. Underscore (\_) should be used between each word of the name. For example, monitor\_settings.

Coding Style

*Column placement****:***

Lines are kept within 120 characters and characters after that should be wrapped accordingly

*Indentation****:***

Indentations, wherever appropriate are kept 4 characters. The tab key can also be used to create the indentations

*Import:*

from <modules folder> import <module file>

from run import <object>

import config

from <python package> import <property>

import <python package> as <var>

import <python package>

import logging

One blank line spacing before importing files.

One blank line spacing before importing python packages.

One blank line spacing before importing logging, followed by initializing the logger.

*Initialization:*

One blank line spacing before initializing MongoDB.

One blank line spacing before initializing module classes in alphabetical order.

*Functions:*

Naming is in lowercase, use underscore ( \_ ) if there are more than one word. Avoid unnecessary verbs (e.g. is, are). Aim towards readability.

Two blank lines spacing before the functions.

Two blank lines between each function.

*End of file:*

One blank line at the end of the file

### Routes

Nomenclature

Files should be named after navigation tab.

Coding Style

*Column placement****:***

Lines are kept within 120 characters and characters after that should be wrapped accordingly

*Indentation****:***

Indentations, wherever appropriate are kept 4 characters. The tab key can also be used to create the indentations

*Import:*

from <route location in blueprints folder> import forms

from <route location in functions folder> import \*

from <modules folder> import <module file>

from run import <object>

import config

from <python package> import <property>

import <python package> as <var>

import <python package>

import logging

One blank line spacing before importing files.

One blank line spacing before importing python packages.

One blank line spacing before importing logging, followed by initializing the logger.

One blank line spacing before initializing MongoDB.

One blank line spacing before initializing blueprint.

One blank line spacing before initializing module classes in alphabetical order.

*Route:*

Have the naming of the route similar to the template name

Naming should aim towards readability and understanding easily

*End of file:*

One blank line at the end of the file

## JAVASCRIPT

Nomenclature

Files are named after their navigation tabs

Coding Style

*Column placement****:***

Lines are kept within 120 characters and characters after that should be wrapped accordingly

*Indentation****:***

Indentations, wherever appropriate are kept 4 characters. The tab key can also be used to create the indentations

*Functions:*

Function names are named using camelCase

All names start with a letter

End all simple statements with a semicolon ( ; )

Always put spaces around operators ( = + - \* / ), and after commas

Do not use addEventListener() to initialize the function because the file shares between different documents. Instead, initialize the functions needed within the document

*Variables:*

Variables are named using camelCase

All names start with a letter

*Comments:*

As the files are grouped within the navigation tabs, one JS file is shared between the templates of the same tab. Comments are used to explain where the particular function is used and the purpose of the function.

**Script Description.** At the beginning of the script, add docstrings to describe the project title, the functions in a single line and the author’s name. Leave one line spacing after the block comment.

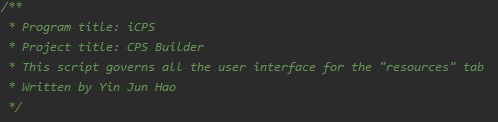


Figure Example of script description for JavaScript.

**Function Description.** Before the function, add comments to describe the overview of the function. Add it before the function for aesthetic reasons.

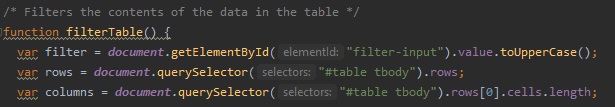


Figure Example of function description for JavaScript.

**Line Description.** Use of single line comment before any logic blocks that are difficult to apprehend at the first look. The comment occupies one line due to aesthetic reasons.



Figure Example of line description for JavaScript.

*End of file:*

One blank line at the end of the file

## CSS

Nomenclature

Files are named after their navigation tabs

Coding Style

*Column placement****:***

Lines are kept within 120 characters and characters after that should be wrapped accordingly

*Indentation****:***

Indentations, wherever appropriate are kept 4 characters. The tab key can also be used to create the indentations

*Property:*

Preferably use of hex color codes (#000) unless for the extremes (black & white)

Put spaces after property ( : ) declarations

Do not indent blocks

Use one line per property declaration

*Rule:*

Put spaces after rule ( { ) declarations

One blank line after every rule declaration.

*Comments:*

As the files are grouped within the navigation tabs, one CSS file is shared between the templates within the same tab. Thus, comments are used to separate the rules used for each template.

**Script Description.** At the beginning of the script, add comments to describe the project title, the use of the stylesheet and the author’s name. Leave one line spacing after the block comment.

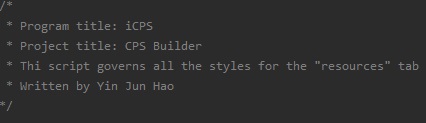


Figure Example of script description for CSS.

**Segment Description**. At the beginning of the segment, add comments to the blocks of rules for each template.



Figure Example of segment description for CSS.

*End of file:*

One blank line at the end of the file

## HTML5

*Templates Component*

Nomenclature

Files are named according to their functions and naming is best kept at singular. Otherwise, the use of underscore ( \_ ) is needed to separate the terms in the name.

Coding Style

*Column placement****:***

Lines are kept within 120 characters and characters after that should be wrapped accordingly

*Indentation****:***

Indentations, wherever appropriate are kept 4 characters. The tab key can also be used to create the indentations

*Classes and ID*:

Lower case is used

Naming is best kept at singular otherwise, a hyphen ( - ) is used to separate the terms in the name.

Spaces are not needed after operators

*Attributes and Elements:*

Lower case is used

*Line Spacing:*

Blank lines are not necessary for the template files.

# CODE

For the full list of code, refer to the following repository in GitHub:

<https://github.com/mpecck/IntelligentCPS>