

# NexusWide Conventions And Regulations Guidelines

Nima Bavar

March 29, 2024



## ©opyright

The corresponding documentation literature and all the containment's, are certified under the CREATIVE COMMONS ZERO v1.0 UNIVERSAL LICENSE , with the elucidation of the actuality that all branches of MODIFICATION, DISTRIBUTION, PATENT, COMMERCIAL and PRIVATE USAGE are entirely authorized.

This fragment of dossier is submitted *AS IS*, you are liberate to manufacture any category of modifications you find essential for your individuality or organization.

However, this aforementioned fact loose the NEXUSWIDE team from any fork of provisioned liability or warranty regarding the usage scenario of such composition, denoting that we are not held accountable for whichever harm caused by the employment/misemployment of this attestation in any form, therefore we demand you to cautiously study the above-mentioned license sanctions if the decision for calibrating this file in a personal or officially concealed environment was made.

The NEXUSWIDE research team welcomes all enhancement ideations announced or pushed to the official repository of the imminent writing.

Dedicated to all the computation enthusiasts  
and  
FREE AND OPEN SOURCE SOFTWARE producers around the globe.

---

# Contents

<b>1</b>	<b>Conventions</b>	<b>1</b>
<b>2</b>	<b>Preface</b>	<b>2</b>
<b>3</b>	<b>Project Management Guidelines</b>	<b>3</b>
3.1	Preamble . . . . .	3
3.2	List of Rules . . . . .	4
3.3	Examples . . . . .	14
<b>4</b>	<b>Designing Algorithm Guidelines</b>	<b>15</b>
4.1	Preamble . . . . .	15
4.2	List of Rules . . . . .	16
4.3	Examples . . . . .	18
<b>5</b>	<b>Programming Guidelines</b>	<b>19</b>
5.1	Preamble . . . . .	19
5.2	List of Rules . . . . .	21
5.3	Examples . . . . .	22
<b>6</b>	<b>File System Management Guidelines</b>	<b>24</b>
6.1	Preamble . . . . .	24
6.2	List of Rules . . . . .	26
6.3	Examples . . . . .	27
<b>7</b>	<b>Markdown Language Guidelines</b>	<b>28</b>
7.1	Preamble . . . . .	28
7.2	List of Rules . . . . .	30
7.3	Examples . . . . .	32

---

<b>8</b>	<b>Information Organizing Guidelines</b>	<b>33</b>
8.1	Preamble . . . . .	33
8.2	List of Rules . . . . .	34
8.3	Examples . . . . .	36

# 1 Conventions

Numerous conventions are globally used throughout this document in order to draw attention or to cite particular meanings, which are stated in the posterior table:

Emphasization		definition
	Text with Tribar	Nexusrule
> <u>ex</u> :: ( <b>Red bold text with angle sign</b> )		Rule example
SMALL CAPS TEXT		Name
<i>Italic text</i>		Important Signification

The acknowledgment of the aforementioned elements shall be enough for you to be prepared to start reading this documentation.

Do not hesitate to reach out to this table anytime you have forgave the reason behind a specific emphasizing regularity.

## 2 Preface

DURABILITY and CONSISTENCY are the two most prominent chunk of a bested project management course, and no project is capable of achieving satisfactory outcomes without proper administration.

One branch of behavior that can further advance durability by obligating consistency in the best feasible manner, is including consequential regulations and laws, which is preserved by affixing sharp style guides to the association.

Thereby to achieve pleasing result in order to avail the esteem and sagacious community of FOSS<sup>1</sup> with our works of knowledge and researches, and most essentially, provide value to our respected readers and users, we shall possess our own particular style guides for the contributors to adhere.

The foundation of the following pertinent standards and rules, was gathered by the acknowledgment of the fore-mentioned principles.

We *oblige* all of our contributors to adhere to them, in order to acquire accommodate flaw-free products.

Please reach out to the succeeding pages, and interpret them precisely.

---

<sup>1</sup>Free and Open and Source Software.

## 3 Project Management Guidelines

### 3.1 Preamble

Comprised throughout numerous well-known projects of the present era, DVCS<sup>2</sup> is one of most comfortable branches of VCS<sup>3</sup>s.

Because of its distributed bearing, the altering of the project is inexpensively feasible in the offline status.

This actuality stems from the fact that each individual is permitted to conduct his\her own local copy of the project, and as labeled in the realm of VERSION CONTROL, *push* the developed REVISIONS<sup>4</sup> to a central cloud repository once a satisfied condition of the project is prevailed.

Thence, it is the default VCS which is utilized by the NexusWide organization in the field of project management.

This section elucidates all of the guidelines and practices which are employed throughout the organization in order to take full advantage of the DVCS capabilities.

---

<sup>2</sup>Distributed Version Control System.

<sup>3</sup>Version Control System.

<sup>4</sup>Also known as Commits.



### 3.2 List of Rules



All NEXUSWIDE project repositories MUST contain a DOCS directory.



All NEXUSWIDE project repositories MUST have a README.MD file located in their DOCS directory which follows the Markdown formatting and style guidelines.



All NEXUSWIDE project repositories MUST contain a SECURITY POLICY file.



All of the following segments: SECURITY POLICY, CODE OF CONDUCT, CONTRIBUTING GUIDELINES, and TERMS OF SERVICE MUST be located within the DOCS directory.



Only the LICENSE file is permitted to be located in the main directory of the repository.



All NEXUSWIDE owned repositories MUST be licensed under ( Creative Commons Zero v1.0 Universal ).



Force commits are not allowed in NEXUSWIDE repositories.



Fast-forward MERGES are not allowed in NEXUSWIDE repositories.



All versions of the project repository MUST be controlled under semantic versioning ( [HTTPS://SEMVER.ORG/](https://semver.org/) ).



The version numbering of the projects **MUST** be split into two phases:  
( pre\_release phase, release phase ).



During the pre release phase all changes in the development process are a part of the version numbering.



The release version numbering **MUST** govern only the user side features.



When a project reaches the release phase, a ( changelog.md ) file **MUST** be implemented in the repository ( docs ) directory.



The release phase version numbering is only mentioned in the  
( changelog.md ) file.





Release versioning and development versioning are separate concepts, do not attempt to increment the release version number while proposing updates to the back end side of the projects.





Local branches **MUST** have their name set as the part of the API they are going to affect, followed by the branch number ( The branch number must be used in order to avoid conflicts with the previous remote branches of the same name ).


> ex : : ( **documentations1** )


 Contributors **MUST** always create a `PULL REQUEST` and branch before attempting to make their local changes and submit an update.


 Remote branches **MUST** be deleted after they have been `MERGED` into the master ( `Main.Project` ) branch in any form.


 `PULL REQUEST` titles **MUST** refer to the `fix`, `feat` or `MAJOR` change commit message which is expected to be the outcome after all the `ISSUE` to-do tasks are finished.

 The convention of naming `PULL REQUESTS` is the same as commit messages.

 All `PULL REQUESTS` **MUST** have at least one comment before being `MERGED`.

 All `PULL REQUESTS` **MUST** contain a well formatted description comment.

 The `PULL REQUEST` description comment **MUST** always be the first comment.

 Contributors **MUST NOT** directly list any of the ( `fix` ) or ( `feat` ) changes in the description comment of a `PULL REQUEST` ( using bulleted or numbered lists, ETC ) They **MUST** be self explanatory in the commit messages.



PULL REQUESTs MUST NOT contain a to do list, as that would make the existence of ISSUES purposeless.



The PULL REQUEST description comment MUST start with a header containing the update name tag

( this is not the same as the PULL REQUEST title, it is any name tag that you would want to assign ) followed by the version number.

*) e.x. : ( Presuming changes in documentations: UDock | 1.0.0 )*



The PULL REQUEST description comment MUST contain four sections:

What has changed ( API changes )?

What were the reasons behind the changes?

What results are expected?

Which ISSUE notes are affected by this update?



The SIGNED OFF BY section MUST be the last section of the description comment and refer to the contributors who parted in the update.



ISSUE references in PULL REQUESTs MUST follow the following format:

( ISSUE Note | ISSUE Number ).



If a update has been rejected, The branch related to the update MUST be deleted, the PULL REQUEST MUST be closed and labled as REJECTED

( The description comment MUST stay as is ).



All code **MUST** be reformatted and pass all the tests adjusted by the GITHUB automation flow before being MERGED.



ISSUE names **MUST** follow the corresponding regular expression:  
( ISSUE Title | ISSUE numeral Count ).

> *e.x* : ( **fix(docs): remove harsh rules | 3** )



All ISSUES **MUST** contain at least 1 comment.



The first comment of an ISSUE **MUST** be the details comment.



The ISSUE details comment **MUST** be formatted in induce to providing a response to the following topics:

What is the assignment?

What are the steps to accomplish?

Why is this fixture necessary?

What is the expected outcome?



The ISSUE details comment **MUST** contain a contributor name at its last section, in order to assign the contributor to the cited task.



The GIT commit messages are formatted by the  
<https://www.conventionalcommits.org/en/v1.0.37>. conventions.



GIT commits MUST be as small and independent as possible, do not concatenate two possible commits together.

> *e.x* : ( **fix(docs): refactor grammar and add header | is a bad commit message.** )



Push changes as often as possible, all the changes to the project ( including the ones that are rejected or/and aborted ) MUST be recorded.



All suspended projects ( those that don't receive any new update for a designated period of time ) MUST be archived and followed by a note text at the tail of their README file, elucidating the fact that the project have been archived.



All GIT commit messages MUST have their branch name and the commit number ( the number of times that the contributor have committed to a branch ) as their footer.

> *e.x* : ( **style\_guide: 3** )



In the context of combining branches, use MERGE instead of REBASE.



While performing any type of merging operation, adhere to the same guidelines as commit messages and provide the reason behind the MERGE.

> *e.x* : ( **fetch(merge): fix conflict with origin** )



All remote repository `MERGE` messages **MUST** be the same as the title of their `PULL REQUEST`.



All remote repository `MERGE` messages **MUST** have the URL of their `PULL REQUEST` at their description section.



All `MERGE` messages ( Including the remote ones ) **MUST** have the `MERGE` pseudo-environment count as their footer: ( The number of times that a `MERGE` operation was conducted in the repository )

> *ex* : : ( **merge: 4** )



All NEXUSWIDE repositories MUST have a Kanban board project derived from the NEXUSWIDE main Kanban board template.



The NEXUSWIDE Kanban board ( IN\_PROGRESS and IN\_REVIEW ) columns MUST be limited to one card only.

( Only one update topic is allowed to be worked on at a time. )



The NEXUSWIDE Kanban board ( WONT\_FIX, READY, DONE ) columns MUST NOT introduce any card amount limitations.



All NEXUSWIDE card items MUST be converted to ISSUES when they reach the ( READY ) state.



All NEXUSWIDE Kanban card names MUST adhere to the NEXUSWIDE GIT commit message guidelines.



All NEXUSWIDE Kanban card names MUST be the name of the ISSUE which is going to be produced after the item have reached the ( IN\_PROGRESS ) state, omitting the ISSUE number.





The ISSUES generated from a Kanban card MUST have the same name as the card.



All NEXUSWIDE Kanban cards MUST have a description section aligned.



All NEXUSWIDE Kanban card descriptions MUST adhere to the ISSUE details comment formatting guidelines.



All NEXUSWIDE Kanban card descriptions MUST be the description of the ISSUE which is going to be produced after the item have reached the ( IN\_PROGRESS ) state.



The ISSUES generated from a Kanban card MUST have the same details comment content as the card description.



All NEXUSWIDE Kanban card items MUST be moved to the ( DONE ) column after the corresponding PULL REQUEST have been MERGED.



All NEXUSWIDE Kanban card items MUST be moved to the ( WONT\_FIX ) column if the update have been rejected.



All NEXUSWIDE Kanban card items after/on the ( READY ) column MUST be assigned a start date.



All NEXUSWIDE Kanban card items after/on the ( READY ) column MUST be assigned a due date.



All NEXUSWIDE Kanban card items after/on the ( READY ) column MUST be assigned to a contributor ( assignee ).



All NEXUSWIDE Kanban card items after/on the ( READY ) column MUST be prioritized and labeled based on the tags offered by the NEXUSWIDE main Kanban template.



All NexusWide Kanban card items after/on the ( READY ) column MUST be measured in size and labeled based on the tags offered by the NexusWide main Kanban template.

### 3.3 Examples

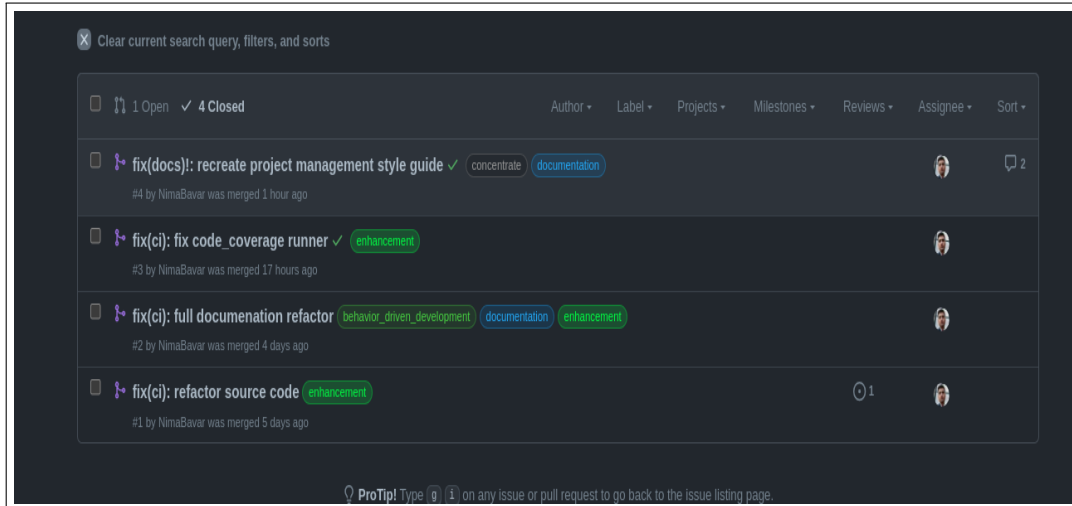


Figure 1: Example of NEXUSWIDE name conventions.

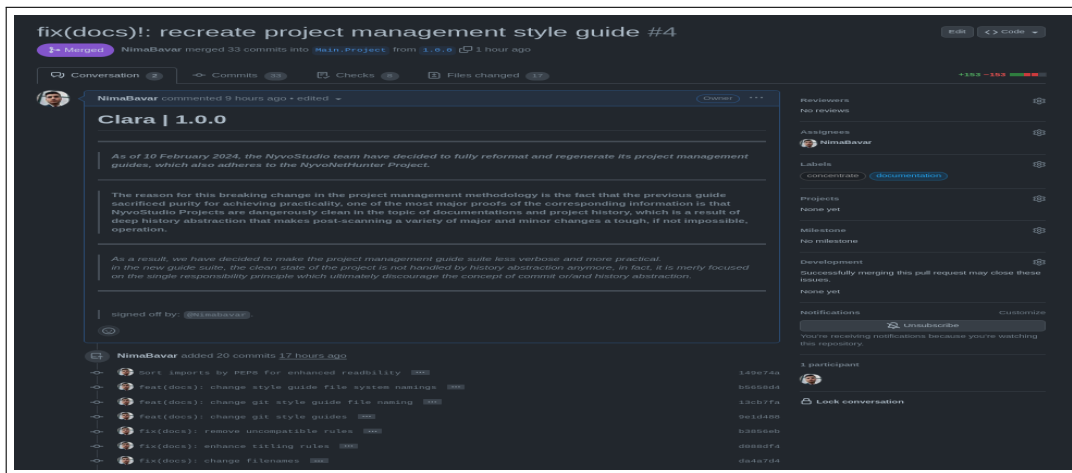


Figure 2: Example of NEXUSWIDE description comment conventions.

## 4 Designing Algorithm Guidelines

### 4.1 Preamble

ALGORITHMS visualize the workflow of a computational process, they are the roadmap to the long-lasting periodic provision of a piece of computational mechanism, in this field, they are comparable to the base-foundational map of a tower engineering project, which easily demonstrates the importance of ALGORITHMS for a *engineering product*.

Surprisingly, the eminent hitherto of validating zero-Vulnerability ALGORITHM conventions, is far beyond the area of communications, as their existence fully rehabilitate the mediocrity of RED-GREEN refactor, in which the correct methodology for solving a problem, is found through exponential trial and error.

Therefore the existence of high-quality ALGORITHMS is capable of redirecting the invested time on experiments, to true and invaluable enhancements.

As every project follows the roadmap it is assigned to, then if the roadmap of the project is volatile, no outcome but failure is expected.

We courageously demand you to elucidate the ALGORITHM conventions of the NEXUSWIDE group so that we can guarantee the assessment of our products.

## 4.2 List of Rules



Class attributes **MUST** be visualized using the attribute shape.



Loops starting, finishing and changes in operations **MUST** be visualized in Square.



Normal logical activities **MUST** be visualized in milestone plan item ( not classes, functions, decorators ).



User inputs **MUST** be visualized in Hexagon shape.



System decisions and exceptions **MUST** be visualized in Rhombus.



Functions **MUST** be visualized in Triangles.



Function calls **MUST** be visualized in call activity shape ( yellow Square ).



Connected attributes **MUST** be visualized in page-references shape.













Class and abstract connections **MUST** be visualized in the class shape.



Databases **MUST** be visualized using the database shape.



Connections **MUST** be visualized in bullet-head lines.

-  Imports, accesses and inherits **MUST** be visualized in arrowhead lines.
-  Partitions **MUST** be visualized in vertical lines.
-  Partition names **MUST** be visualized in the case file item.
-  Comments **MUST** be visualized in comment alt shape.
-  Numerical orderings **MUST** be visualized in brackets.
-  Implement indications are visualized in implementation event.
-  All algorithm files ( excluding the text-based ones ) **MUST** contain a diagram info object explaining the purpose of the file.
-  The diagram info object **MUST** contain an ( end ) object, describing the outcome or the exportation of the algorithm.
-  All textual subjects **MUST** be written using the JetBrains Sans Mono font family.
-  All algorithms **MUST** be visualized in Visual Paradigm.

### 4.3 Examples

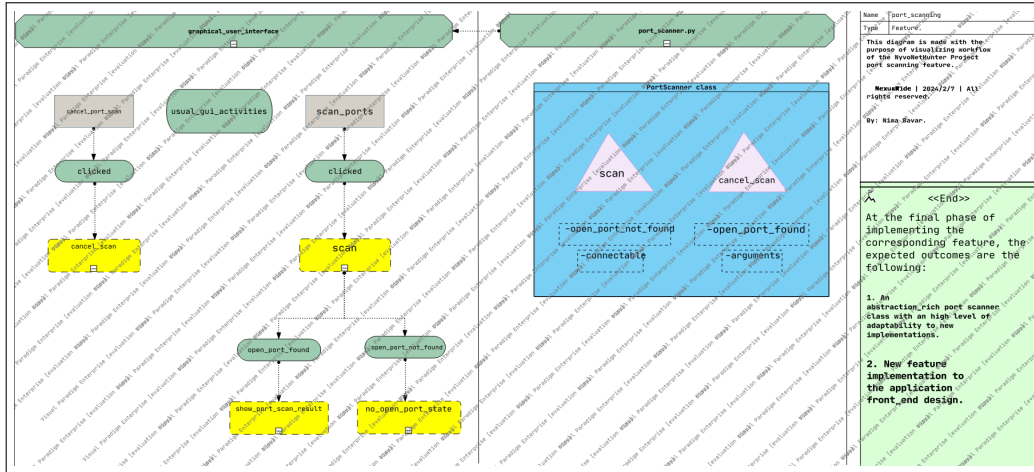


Figure 3: Example of NEXUSWIDE ALGORITHM design conventions.

## 5 Programming Guidelines

### 5.1 Preamble

Due to the modular form of a multifarious amount of programming languages, including `LATEX` and `PYTHON`<sup>5</sup> several people are enabled to work on the same application or feature concurrently, each with their own considerations and mentality.

in such circumstances, the absence of a comparable programming style would result in numerous time consuming conversations and courses, in order to apprehend the workflow of a codebase with differentiations at each of its sections.

This operation consumes the invaluable time which could be invested into effectively *reviewing* and *enhancing* the codebase, therefore results in insufficient outcomes.

The above-mentioned illustrations are the results of the lack of consistent regulations between the individuals thus they become even more prominent when a large number of entities are involved in a project and tasks are broken into countless fragments.

---

<sup>5</sup>Languages that are generally used in the NexusWide corporation for generating documentation files and automating tasks.



Fortunately, all of the aforementioned dilemmas can be prevented by decreasing the amount of creativity arrayed into the programming style of each person by developing a default guide for the populace in the organization to adhere, so that amount of creativity can be directed into the actual development process of the working industry.

By endorsing this accuracy, we have conducted our own programming style guides with the aim of avoiding the referred considerations, which can be read at the following pages.

## 5.2 List of Rules



All programmers who use PYTHON, MUST follow the PEP8<sup>6</sup> style guide.



Programmers who use other languages, MUST follow their language style guide, but still adhere to the blank line rules of PEP8.



All source code documentations must follow the NUMPY docstring formatting guidelines.



All codes MUST be reformatted by PSF/BLACK<sup>7</sup> before being merged to the master ( MAIN.PROJECT ) branch.



All codes MUST have *at least* 80 percent of unit test coverage acceptance rate before being merged to the master ( MAIN.PROJECT ) branch.



All unit test codes must have at *at least* 80 percent of mutation test acceptance rate before being merged to the master ( MAIN.PROJECT ) branch.



All NEXUSWIDE L<sup>A</sup>T<sub>E</sub>X documents MUST contain the NEXUSWIDE logo in their cover or/and title page.

---

<sup>6</sup>Python Enhancement Proposal.

<sup>7</sup>A code analyzer and formatter written for Python.

### 5.3 Examples

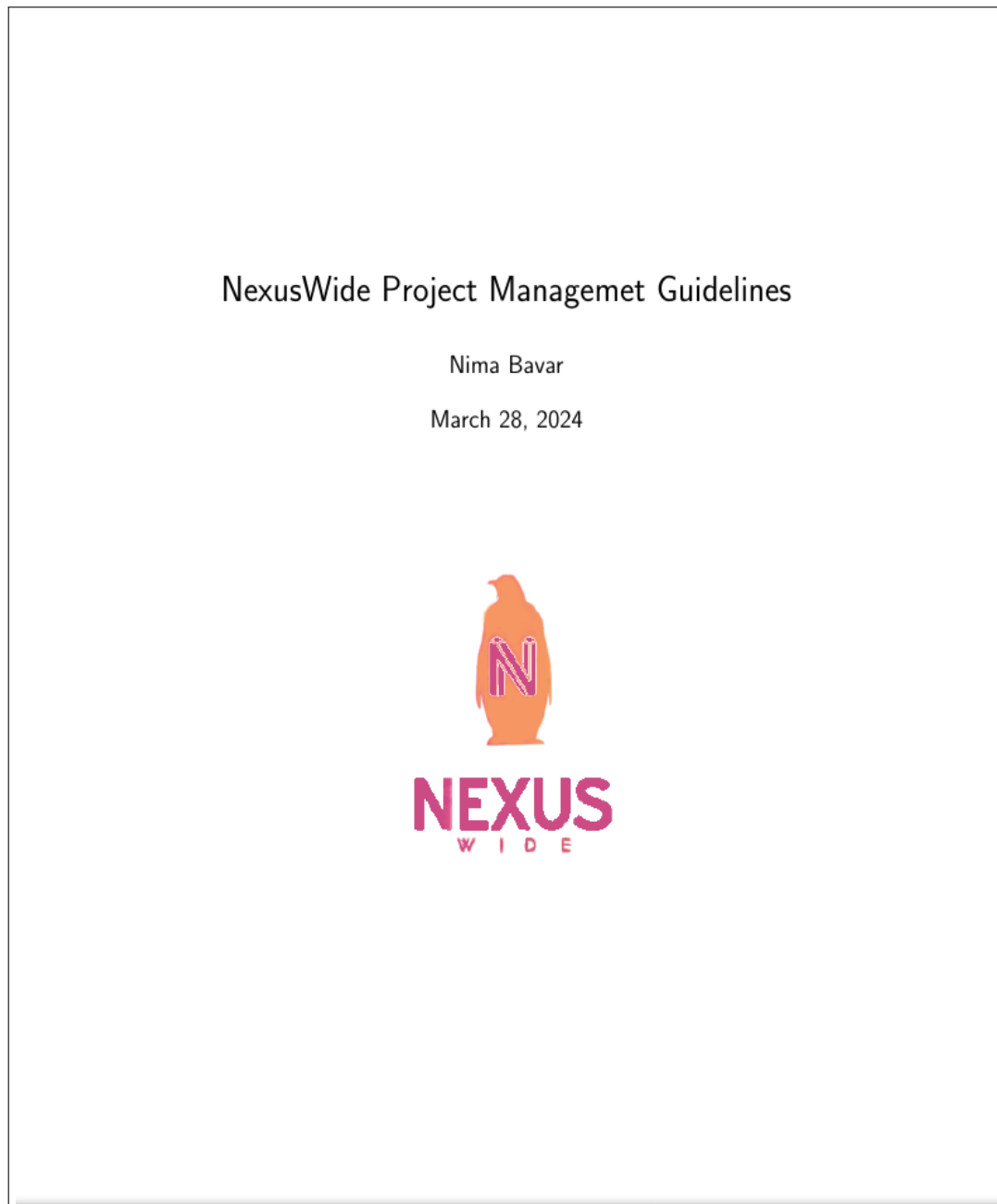


Figure 4: Example of NEXUSWIDE L<sup>A</sup>T<sub>E</sub>X documentation file title page.

```

133 |         raise NoRunningSession("Cannot attempt to cancel a scan while none is running.")
134 |
135 |     self._requested_cancel_scan = True
136 |
137 | def get_scan_data(self, data: Literal["open_ports"] = "open_ports") -> list:
138 |     """
139 |     Retrieves a specific data from the scan result.
140 |
141 |     Parameters
142 |     -----
143 |     data : { "open_ports" }, defaults="open_ports"
144 |         The data to grep from the scan.
145 |
146 |     Returns
147 |     -----
148 |     list
149 |         The data look_up result.
150 |
151 |     Raises
152 |     -----
153 |     NoScanHistoryError
154 |         If the look_up was attempted before a valid scanself.
155 |     TypeError
156 |         If the provided data is not valid.
157 |     """
158 |     valid_datas = ["open_ports"]
159 |
160 |     successful_scan_not_occured = self._scan_attempts == 0 or not self._bool_scan_finished or not self._scanner.rc == 0
161 |     if successful_scan_not_occured:
162 |         raise NoScanHistoryError("Please scan at least 1 host in order to receive the open ports.")
163 |
164 |     if not data in valid_datas:
165 |         raise TypeError(f"{data} is not a valid/or an existing scan data.")
166 |
167 |     hosts = [host for host in self.scan_result.hosts]
168 |
169 |     if data == "open_ports":
170 |         open_ports = [f'{host.ipv4}: {host.get_open_ports()}' if host.is_up() else f'{host.ipv4}: host is down.' for host in hosts]
171 |         retrieved_scan_data = open_ports
172 |
173 |     return retrieved_scan_data
174 |
175 |

```

Figure 5: Example of NEXUSWIDE PYTHON programming style.

## 6 File System Management Guidelines

### 6.1 Preamble

Since the dawn of technological computation, RESOURCE MANAGEMENT have existed, and one of the most eminent subjects discussed in this field, is the persistent nature of the resources.

Fortunately, due to the existence of PERIPHERAL MEMORY SYSTEMS<sup>8</sup> ( As instance HDD<sup>9</sup>, SSD<sup>10</sup>, ETC ) at the present epoch, we are not obliged to face the remnant flaw of grasping a solution for producing persistent file systems.

However, we are responsible for finding solutions for easing the process of navigating the file systems that we generate, therefore, we can either make finding items in a data storage a long-lasting and preferably bothering memory, or make it as suitable as possible for our own comfort.

The assurance of the fact that many individuals have faced filenames with labels alike ( test.something, popcorn ),

and in rapidly worse scenarios, ( dont' touch this file ) is a completely acknowledged actuality.

Resources, files and directory tags are similar to names for a millennial, therefore they shall include the use case and the reason for their own existence.

---

<sup>8</sup>Persistent and large storage memories.

<sup>9</sup>Hard Disk Drive, a type of sequential access storage systems.

<sup>10</sup>Solid State Drive, a branch of direct access storage systems.

Similar to humans which have incomparable preferences in naming their children, this state does again visualize itself while choosing tags for file system elements, and this acknowledgment may make navigating file systems made by other entities, a uncomfortable period of navigation.

We have decided to conduct a set of default morals for handling such situations, to evade the encountering annoying statuses, all of which can be read at the following pages.

## 6.2 List of Rules



Only the `ENGLISH` letter Characters `[A-Z]` and numbers `[0-9]` are allowed to be used in file names.



All file names **MUST** be in `SNAKE_CASE`: ( All letters **MUST** be in lowercase, instead of space underscore(`_`) is used ).



File names **MUST** start with a verb, and at the second part there **MUST** be a name or piece of information.



File names **MUST** be descriptive and self explanatory.

## 6.3 Examples

```
/home/nimabavar/Desktop/work/diary/projects/nexus_wide/repositories/style_guide/attachments:  
total used in directory 52K available 181.9 GiB  
drwxr-xr-x 2 nimabavar nimabavar 4.0K Mar 28 13:46 designing_algorithm_style  
drwxr-xr-x 2 nimabavar nimabavar 4.0K Mar 29 06:58 markdown_language_style  
drwxr-xr-x 2 nimabavar nimabavar 4.0K Mar 28 13:46 organizing_file_style  
drwxr-xr-x 2 nimabavar nimabavar 4.0K Mar 28 13:46 organizing_information_style  
drwxr-xr-x 2 nimabavar nimabavar 4.0K Mar 28 13:46 programming_style  
drwxr-xr-x 2 nimabavar nimabavar 4.0K Mar 28 13:46 project_management_style  
-rwxr-xr-x 1 nimabavar nimabavar 20K Mar 28 13:46 nexus_wide_logo.png
```

Figure 6: Example of NEXUSWIDE file system management style.



## 7 Markdown Language Guidelines

### 7.1 Preamble

DIRECT COMMUNICATION is a trait of the beings nature which preserve deep roots into the definition of the word, HUMAN.

Although communication have always existed, it has also evolved through the passage of the time, from the era of the cavemen and their paintings, until this moment, in which you are reading this documentation through a LIQUID CRYSTAL or CATHODE RAY TUBE monitor.

Disregarding the state of the improvised enhancements throughout the years, one shared element have always been the same in the heart of every communication, and that is, COMMON UNDERSTANDING.

This COMMON UNDERSTANDING was obtained by numerous methodologies, including SHAPES, LANGUAGES, LITERATURE, TYPEFACES, FONTS and currently, MARKUPS.

Just as a rich speaking language with a humble accent, can flow and direct the words and mentalities of a speaker comfortably, a apprehended set of MARKUP TAGS are capable of achieving the same results.

There is only one unsatisfactory acceptance that we have to make in order to acquire COMMON UNDERSTANDING, and it is that both of the speakers and messengers, shall speak the same language.

And this rule also applies to our Brave New SILLICON VALLEY alike communications, indicating that if we do not speak the same language as each other, we would have to invest superior hours, explaining the foundation of our language.

The solution to this problem, is yet known, and it is that for all humans around the globe, to speak the same language.

g In order to decrease the happening of communication degrades in our own organization, we have also produced rules and guidelines for our spoken language, THE MARKDOWN DOCUMENTATION FORMAT, which can be read at the following pages.

## 7.2 List of Rules



Every markdown file **MUST** contain at least one header.



Headers should not be **bold**, *italic*, ETC, they **MUST** be displayed in simple text.



All headers **MUST** be the highest priority header ( # ), Usage of other headers is not allowed.



All ENGLISH language sentence rules **MUST** be applied to headers

> e.x. : ( **Sentence **MUST** start with a uppercase letter** ).

> e.x. : ( **My name is John Doe.** )



All headers except for the table of content, **MUST** be a question sentence.



The header texts **MUST** be simple and descriptive, do not use complex words.



Every header **MUST** contain a text after ( inside ) answering the question that was asked in the header, which is acknowledged as subheader.



subheaders **MUST** answer the questions with a comprehensive and plain answer.



subheaders should never contain extra sentences other than the ones that are answering the question.



All subheader texts must come after a ruler line, which is inserted using the `---` command in `MARKDOWN`.



All subheader texts must end a ruler line, which is inserted using the `---` command in `MARKDOWN`.



All subheader text lines must start with the `>` formatting character.

## 7.3 Examples

### # What are conventions used throughout this project?

By any situation , All the employees of this project must follow the [styling idioms of NexusWide](#) which are:

- File system structure style.
- Text organizing style.
- Algorithm designing style.
- Markdown language style.
- Programming style.

- **Project management style.**

### # How can we contact the organization authorities for submitting a report?

To report any possible security vulnerability, legal concern, harassment, or misguidance of code of conduct by an employee, we kindly request that you reach out to the NexusWide team through the following methods:

- NexusWide mail box : [nexuswide.contact@protonmail.com](mailto:nexuswide.contact@protonmail.com)
- **Reporting on this repository by [Creating a new issue](#)**

Figure 7: Example of NEXUSWIDE MARKDOWN language formatting style.

## 8 Information Organizing Guidelines

### 8.1 Preamble

We have annotated multifarious human preference related topics in the previous segments of the document, and have noted that to avoid disruption in the development process of a certain application, we *MUST* avoid interpolating our own preferences for labeling objects in the environment.

It is not only the label of the objects which is the matter of evaluation, but also their content, as it doesn't matter how meaningful the label of a `BINARY CODE`<sup>11</sup> file is, the workflow of its content is yet unknown to a human being.

This matter also includes human-written text and raw ASCII<sup>12</sup> files, as most of the time while reading and interpreting contents written by another person, due to their preference in the way of representing intelligence, they may be totally unreadable to us.

So we have injected the same fixture that we have included while discussing the clarity of labels in a file system, and that is, the preparation of a consistent and durable set of guidelines which can be read at the following pages.

---

<sup>11</sup>A type of coding system used in Computation sciences including Boolean Algebra and modern Technological Computation.

<sup>12</sup>American Standard Characters for Information Interchange , A set of characters used mainly in the Internet for transferring messages.

## 8.2 List of Rules



Each section of a text file should be partitioned using `TITLE` texts.



All `TITLES` **MUST** be separated by two blank lines.



`TITLE` names **MUST** be descriptive and simple, do not introduce redundant complexity.



Each `TITLE` is allowed to contain numerous `SUBTITLES`.



All `SUBTITLES` **MUST** be used only inside a `TITLE`.



All first `SUBTITLES` inside **MUST** be separated from the `TITLE` by one blank line.



All `SUBTITLES` **MUST** be one blank line separated from each other.



All `SUBTITLES` **MUST** have four indentation ( space character ) levels before their main text.



All `SUBTITLES` **MUST** follow a incrementing numbering system.



All `SUBTITLE` numbers **MUST** be in English.



All SUBTITLES MUST have a dot(.) after their numbering.



All SUBTITLES MUST have one space between their dot and the main text.



All SUBTITLES MUST end with a punctuation mark.



All of the text files MUST end with a ---- END ---- title tag.



## 8.3 Examples



```
main_notes.txt
1 ---- project_changelog ----
2
3
4 back_end_implementations:
5
6 1. Added the port scanner class.
7
8 2. Added ( url_suffix_removal ) feature to the ( NyvoNetHunterUrl ) class.
9
10 3. Added ( url_path_removal ) feature to the ( NyvoNetHunterUrl ) class.
11
12 4. Added the new ( port_scanner_exceptions ) exception suite.
13
14
15 back_end_patches:
16
17 1. Removed the unnecessary ( UnexpectedArgumentTypeError ) exception.
18
19
20 graphical_user_interface:
21
22 1. Refactored the graphical user interface element placements.
23
24
25 source_code_patches:
26
27 1. Enhanced source_code_documentations.
28
29 2. Refactored function signatures.
30
31
32 file_system_patches:
33
34 1. Refactored the file locations.
35
36 2. Refactored the file system module namings.
37
38
39 ---- END ----
```

Figure 8: Example of NEXUSWIDE information organization conventions.