Software Design & Development

Major Project

Requirements and Design Report

THE Z PROGRAMMING LANGUAGE

Oliver Lenehan

Term One, 2020 V1.0

Table of Contents

[x] Introduction

The following report documents the requirements and design of the
`Z Programming Language`
and development environment.

Included in the requirements report:
The Client's needs
Business requirements fulfilling the Client's needs
Technical requirements
Representation of data processes
Project plan timeline

Included in the design report:
Algorithms
Graphical Design Illustrations
Document of Data Items.

[x] Client Needs

The client needs for the Z Programming Language environment are:

- A **compiler**, taking Z source code and representing it in **python** 'object code'.
- Perform Lexical and Syntax analysis on Z source code.
- **Error messages** displayed during Lexical and Syntax Analysis.

The client needs for the Z Programming Language itself are:

- Control Structures:
  - Programs

    -
      ```
      PROG procedurename
        instruction:
        instruction:
      ENDPROG
      ```
  - Instruction Sequences
    - Where each instruction ends (or is separated by) a colon (:).
  - Loops
    - Pre-Test

      -
        ```
        WHEN condition(s)
          DO instructions:
          ENDDO
        ENDWHEN
        ```
    - Post-Test

      -
        ```
        DO instructions:
        UNTIL condition
        ENDDO
        ```
    - Count

      -
        ```
        FOR variable FROM number TO number BY number
        DO instructions: ENDDO
        ENDFOR
        ```
  - Selection
    - Binary

      -
        ```
        IF conditions(s)
          DO instructions: ENDDO
        OTHERWISE
          DO instructions: ENDDO
        ENDIF
        ```
    - Multiway

      -
        ```
        SWITCH variable WHEN value
          DO instructions ENDDO
        ENDSWITCH
        ```
- Definitions:
  - Maths Operators
    - `+ - / *`
  - Comparison Operators
    - `< (less than)`
    - `> (greater than)`
    - `<= (less than, or equal to)`

        - `>= (greater than, or equal to)`
        - `== (equal to i.e. equality check)`
        - `&& or AND (if both are true, evaluate to TRUE, else FALSE)`
        - `|| or OR (if either are true, evaluate to TRUE, else FALSE)`
        - `! or NOT (!== or !> or !<)`
    - Assignment Operator
        - `=`
    - Variables
        - Must start with a letter, followed by 1 or more digits.
        - May contain any data-type from the Z Programming Language.
        - Example: `= A999 123`
    - Letters
        - All uppercase letters of the English Alphabet, i.e. A...Z
    - Strings
        - Start and End with double quotes (`"`).
        - Contains zero or more letters or digits.
    - Digits
        - `0 1 2 3 4 5 6 7 8 9`
        - `‘.’` is used for floating point numbers, e.g. `3.14`
    - Conditions:
        - Evaluate to TRUE or FALSE
        - Comparison Operator followed by variables or letters or digits.
        - E.g. `== A9 “A”`
        - E.g. `AND == A1 A2 > B1 B2`
- Instructions:
    - Statements
        - Terminate with a “`:`” symbol. (`statement1: statement2:`)
    - Assignment Statements
        - `= A1 B2` (put value of B2 into A1)
        - `= A1 1` [any maths operator] (set A1 equal to A1 (maths operator) 1)
          E.g. `= A1 1 +` (Add 1 to A1)
    - Output Statements
        - `OUT [“some text”]:`
        - `OUT [digits]`
        - `OUT [variable]`
        - `OUT [“Value: ”+ 999 + “ ” + A999]:`
    - Input Statements:
        - `= A9 IN [“prompt: ”]:`

Requirements Report

[x] Functional Requirements

To satisfy the Client's needs, the Z Programming Language environment should provide the following functionality.

The System will allow the User to:

1. Receive a Z source program, compile it, and output a python file to be run.
2. Perform syntax and lexical analysis on source code.
3. Present user-friendly messages regarding incorrect keywords or incorrect program structures resulting from syntax and lexical analysis.
4. Present a graphical user interface.
5. Allow for code modification in the GUI.
6. Allow for code compilation through the GUI.
7. Execute compiled Z code from the GUI.
8. Provide user documentation of the Z Programming Language
9. Provide user documentation about the development environment.

[x] Non-Functional Requirements (Technical)

The following is a list of Requirements that do not relate directly to the business function of the systems, such as technical requirements, etc.

1. User Preferences stored in JSON files.
2. GUI provided through a dynamically-generated website, accessible through a browser.
3. Implementation of Web Server and Program Logic
    a. TypeScript Programming Language
    b. Deno JavaScript Runtime (Including Standard Library)
    c. Web Page Scripting (JavaScript)
    d. Integration of Webpage + Server using WebSockets.
4. Development Tools
    a. Visual Studio Code (Code Editing for Pages/Logic)
    b. Deno runTests() functionality for testing.
    c. Custom build script implemented to bundle the program for installation in a "build" folder. (Packaging Deno, Compiled Code, Documentation)
5. Images and Webpage design done from scratch.
6. Original source code, with assistance through standard libraries/functions.
    a. HTTP, FileSystem
7. Using Python to run compiled Z code.

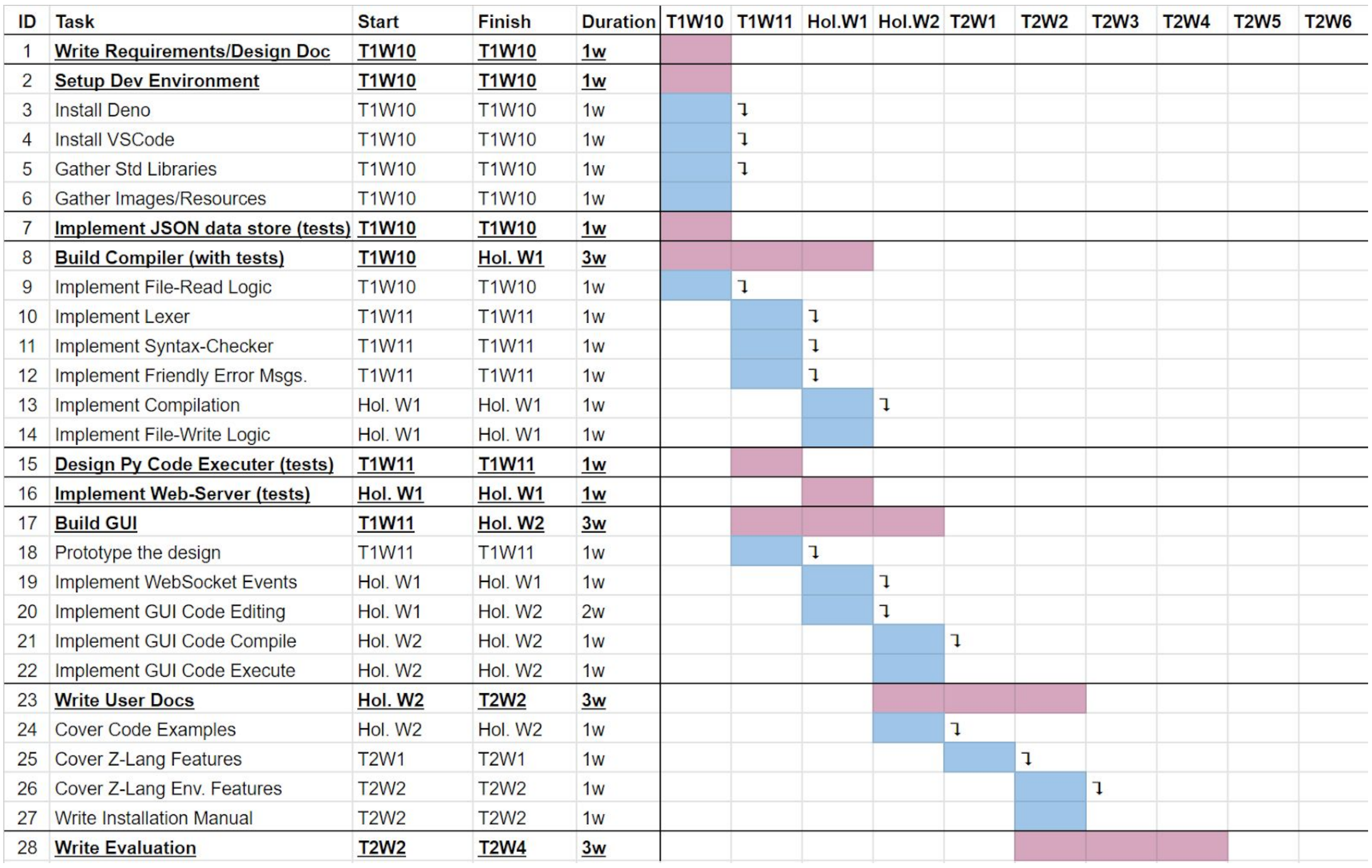
[x] Input-Process-Output Chart

The main Input, Processes and Outputs to satisfy the Functional/Non-Functional Requirements are documented here:

Entity	**Input**	**Process**	**Output**
User	Z Source File	**Compile Z Program** ● Load source code ● Run Lexical Check ● Run Syntax Check ● Compile to Python	Python Object File
User	Z File Location	**Opening Z Source for Editing** ● Load file from location. ● Read source code and display on screen.	Open code editor.
User	Python Object File	**Run a Z Program** ● Call Python Executable with Object file as parameter. ● Pipe python output to screen output.	Display through screen/stdout.
Z-Runtime	Error Code	**Display Error Message** ● Lookup in the error dictionary the message for the code, and display this on-screen.	Display through popup/red message
User	Open Help Event	**Display User Help/Docs** ● Load help/docs ● Output html content to screen.	Display help/doc screen.
Z Web Server	Event data	**Handle WebSocket Event** ● Perform an action (save/open) based on the Event data. E.g. file saved, close file, open file, etc	Sends an ACK.
Z Web Server	Event Data	**Send WebSocket Event** ● Serialise Event Data, e.g. python exec output, file data	Sends Event Data.
User	Preferences	**Apply User Settings** ● **Write preferences to file.**	Update screen to show changed preferences.

[x] Project Plan

The following is a gantt chart that documents the tasks and effort/duration required to complete the task by the required date



ID	Task	Start	Finish	Duration	T1W10	T1W11	Hol.W1	Hol.W2	T2W1	T2W2	T2W3	T2W4	T2W5	T2W6
1	**Write Requirements/Design Doc**	**T1W10**	**T1W10**	**1w**	🟪									
2	**Setup Dev Environment**	**T1W10**	**T1W10**	**1w**	🟪									
3	Install Deno	T1W10	T1W10	1w	🟦	↴								
4	Install VSCode	T1W10	T1W10	1w	🟦	↴								
5	Gather Std Libraries	T1W10	T1W10	1w	🟦	↴								
6	Gather Images/Resources	T1W10	T1W10	1w	🟦									
7	**Implement JSON data store (tests)**	**T1W10**	**T1W10**	**1w**	🟪									
8	**Build Compiler (with tests)**	**T1W10**	**Hol. W1**	**3w**	🟪	🟪	🟪							
9	Implement File-Read Logic	T1W10	T1W10	1w	🟦	↴								
10	Implement Lexer	T1W11	T1W11	1w		🟦	↴							
11	Implement Syntax-Checker	T1W11	T1W11	1w		🟦	↴							
12	Implement Friendly Error Msgs.	T1W11	T1W11	1w		🟦	↴							
13	Implement Compilation	Hol. W1	Hol. W1	1w			🟦	↴						
14	Implement File-Write Logic	Hol. W1	Hol. W1	1w			🟦							
15	**Design Py Code Executer (tests)**	**T1W11**	**T1W11**	**1w**		🟪								
16	**Implement Web-Server (tests)**	**Hol. W1**	**Hol. W1**	**1w**			🟪							
17	**Build GUI**	**T1W11**	**Hol. W2**	**3w**		🟪	🟪	🟪						
18	Prototype the design	T1W11	T1W11	1w		🟦	↴							
19	Implement WebSocket Events	Hol. W1	Hol. W1	1w			🟦	↴						
20	Implement GUI Code Editing	Hol. W1	Hol. W2	2w			🟦	↴						
21	Implement GUI Code Compile	Hol. W2	Hol. W2	1w				🟦	↴					
22	Implement GUI Code Execute	Hol. W2	Hol. W2	1w				🟦						
23	**Write User Docs**	**Hol. W2**	**T2W2**	**3w**				🟪	🟪	🟪				
24	Cover Code Examples	Hol. W2	Hol. W2	1w				🟦	↴					
25	Cover Z-Lang Features	T2W1	T2W1	1w					🟦	↴				
26	Cover Z-Lang Env. Features	T2W2	T2W2	1w						🟦	↴			
27	Write Installation Manual	T2W2	T2W2	1w						🟦				
28	**Write Evaluation**	**T2W2**	**T2W4**	**3w**						🟪	🟪	🟪		

Design Report

The following documents, for each of the Functional Requirements, the Flowchart/Pseudocode Algorithms, Storyboards documenting User Interfaces and any Graphical Elements (images, logos, buttons, etc)

[x] Algorithms

The following are the algorithms documenting the major functions within the Z Programming Language Environment.

```
BEGIN loadSourceCode( filename )
  OPEN filename for READ
  SET filePtr to File
END loadSourceCode

BEGIN lexicalAnalysis( file )
  SET lexList = Array
  SET currLex = String
  WHILE filePtr NOT EOF
    SWITCH filePtr
      CASE appears in string: currLex = currLex + filePtr
      CASE appears in number: currLex = currLex + filePtr
      CASE appears in operators: currLex = currLex + filePtr
      CASE valid keyword: currLex = currLex + filePtr
      OTHERWISE:
        IF filePtr is whitespace THEN
          Add currLex to lexList
          Clear currLex
        ELSE
          ERROR UNEXPECTED TOKEN
        ENDIF
    END SWITCH
    INCREMENT filePtr
  END WHILE
  RETURN lexList
END lexicalAnalysis
```

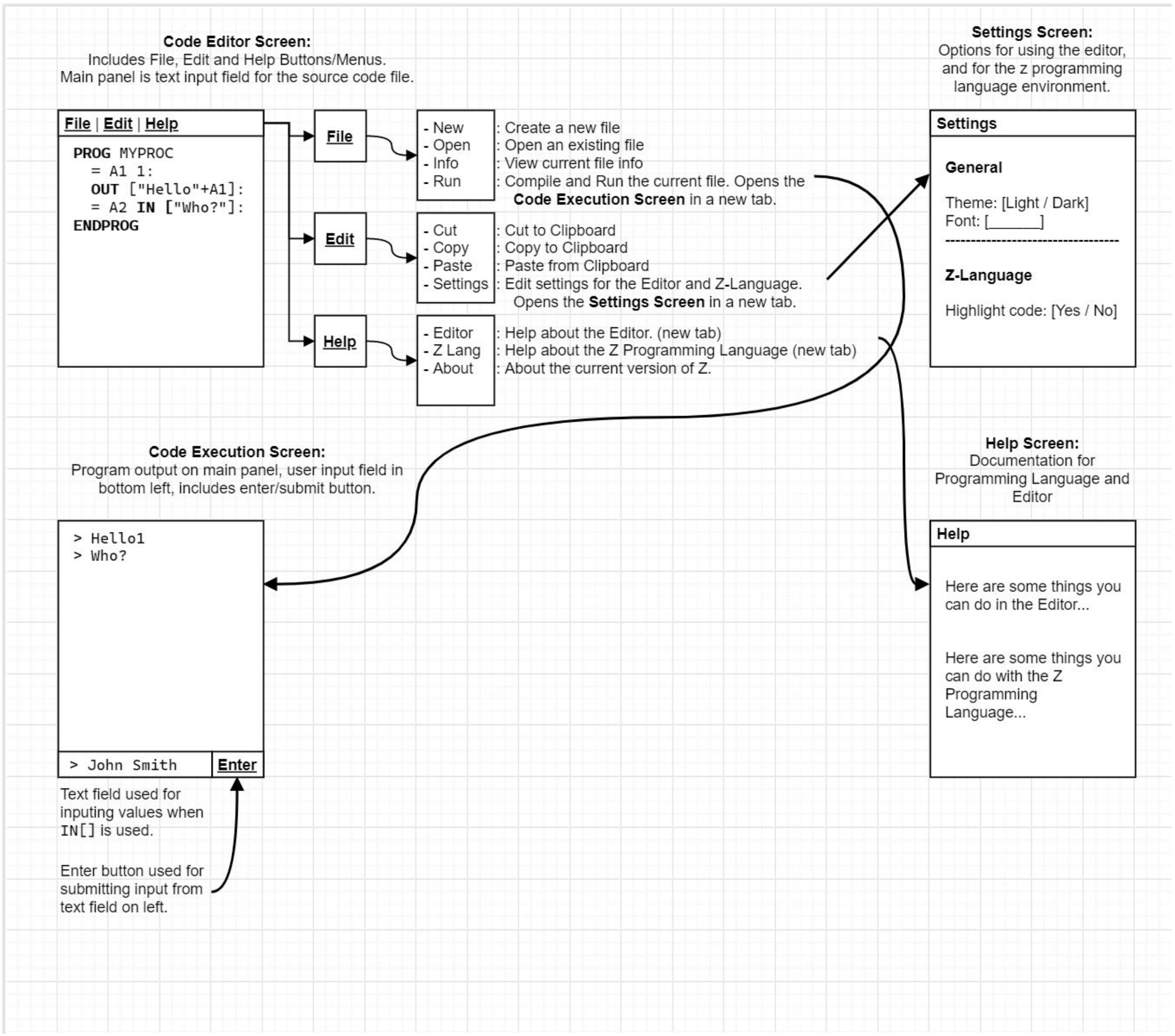
```
BEGIN syntaxAnalysis( lexList )
  abstractSyntaxTree = Record
  FOR each item in lexList
    IF item is a keyword THEN
      Test each following item, and ensure it matches the order specified for
      that keyword. E.g. “OUT” followed by [ output text ]
      If successful, begin building the abstractSyntaxTree branch for that
      structure.
    ELSE
      Begin adding to a stack, the items in sequence. And test that against the
      sequences of conditions and other non-terminals. If success, add to
      abstract syntax tree; on fail, THEN throw ERROR syntax invalid.
    ENDIF
  ENDFOR
  RETURN abstractSyntaxTree
END syntaxAnalysis

BEGIN compileToPython( abstractSyntaxTree )
  OPEN outputFile for WRITE
  FOR each branch on abstractSyntaxTree
    WRITE to outputFile: substituteForPython(branch keyword, branch params)
  ENDFOR
END compileToPython

BEGIN substituteForPython( keyword, param1, param2, etc )
  RETURN outputString
END substituteForPython
```

[x] EBNF for Z Language

```
letter ::=
  (All uppercase English letters, A-Z)

digit ::=
  (All digits, 0-9)

whitespaceChar ::=
  (tab) | (space) | (newline)

whitespace ::=
  <whitespaceChar> {<whitespaceChar>}

mathOperator ::=
  "+" | "-" | "*" | "/"

comparisonOperator ::=
  "<" | ">" | "<=" | ">=" | "==" | "&&" | "||" | "!" | "AND"
| "NOT" | "OR"

integer ::=
  <digit> {<digit>}

float ::=
  <integer> "." <integer>

number ::=
  (<integer>|<float>)

string ::=
  "" {<letter>|<digit>} ""

variable ::=
  <letter> {<digit>}

input ::=
  "IN" [<whitespace>] "[" [<whitespace>] <string>
[<whitespace>] "]"

output ::=
  "OUT" [<whitespace>] "[" [<whitespace>]
(<variable>|<string>|<number>) {[<whitespace>]
"+" [<whitespace>]
(<variable>|<string>|<number>)} [<whitespace>] "]"

assign ::=
  "=" [<whitespace>] <variable> <whitespace>
(<variable>|<string>|<number>|<input>)
[[<whitespace>]<mathOperator>]

statement ::=
  (<assign>|<output>) [<whitespace>] ":"

condition ::=
  <comparisonOperator> <whitespace>
(((<variable>|<string>|<number>) <whitespace>
(<variable>|<string>|<number>)) | <condition>)

preTest ::=
  "WHEN" <whitespace> <condition> <whitespace>
"DO" <whitespace> {<instruction> <whitespace>}
"ENDDO" <whitespace> "ENDWHEN"

postTest ::=
  "DO" <whitespace> {<instruction> <whitespace>}
"UNTIL" <whitespace> <condition> <whitespace>
"ENDDO"

countLoop ::=
  "FOR" <whitespace> <variable> <whitespace>
"FROM" <whitespace> <number> <whitespace>
"TO" <whitespace> <number> <whitespace> "BY"
<whitespace> <number> <whitespace> "DO"
<whitespace> {<instruction> <whitespace>}
"ENDDO" <whitespace> "ENDFOR"

select ::=
  "IF" <whitespace> <condition> <whitespace>
"DO" <whitespace> {<instruction> <whitespace>}
"ENDDO" <whitespace> {"OTHERWISE"
<whitespace> "DO" <whitespace> {<instruction>
<whitespace>} "ENDDO" <whitespace>} "ENDIF"

switch ::=
  "SWITCH" <whitespace> <variable>
<whitespace> "WHEN" <whitespace>
(<string>|<number>) <whitespace> "DO"
<whitespace> {<instruction> <whitespace>}
"ENDDO" <whitespace> "ENDSWITCH"

control ::=
(<preTest>|<postTest>|<countLoop>|<select>|<swi
tch>)

instruction ::=
  <statement> | <control>

procedureName ::=
  <letter> { <letter> }

program ::=
  "PROG" <whitespace> <procedureName>
<whitespace> { <instruction> <whitespace> }
"ENDPROG"
```

[x] Storyboards

The following shows the relationships between pages, and navigation to other pages



Code Editor Screen:
Includes File, Edit and Help Buttons/Menus.
Main panel is text input field for the source code file.

```
PROG MYPROC
  = A1 1:
  OUT ["Hello"+A1]:
  = A2 IN ["Who?"]:
ENDPROG
```

- New	: Create a new file
- Open	: Open an existing file
- Info	: View current file info
- Run	: Compile and Run the current file. Opens the **Code Execution Screen** in a new tab.

- Cut	: Cut to Clipboard
- Copy	: Copy to Clipboard
- Paste	: Paste from Clipboard
- Settings	: Edit settings for the Editor and Z-Language. Opens the **Settings Screen** in a new tab.

- Editor	: Help about the Editor. (new tab)
- Z Lang	: Help about the Z Programming Language (new tab)
- About	: About the current version of Z.

Settings Screen:
Options for using the editor, and for the z programming language environment.

Settings

General

Theme: [Light / Dark]
Font: [______]

Z-Language

Highlight code: [Yes / No]

Code Execution Screen:
Program output on main panel, user input field in bottom left, includes enter/submit button.

```
> Hello1
> Who?
```

```
> John Smith
```

Text field used for inputing values when IN[] is used.

Enter button used for submitting input from text field on left.

Help Screen:
Documentation for Programming Language and Editor

Help

Here are some things you can do in the Editor...

Here are some things you can do with the Z Programming Language...

[x] Graphical Elements

The Z Programming Language uses the following assets in its design...

Z Programming Language Logo
Used as the "banner art" on the editor. App-Icons utilise just the "Z" featured below.

THE
Z
PROGRAMMING
LANGUAGE

[x] Data Dictionary

Item	Type	Format	Storage Size	Display Size	Descript ion	Example	Validatio n
Z Source File	File, Read as TEXT	Z Language	Dynamic	n/a	Source code to be compiled	BEGIN PRO: END	
File Path	TEXT	Path/ Url / Directory	Typically 256 bytes	n/a	Where a file is located.	C:\file.z	
Python Object File	File, ReadWrite as TEXT	Python Language	Dynamic	n/a	Output from the compiler	print()	
Event Data	Record	JSON	Dynamic, Length depending on payload data	n/a	Used between Client/ Server for communic ation	{}	JSON Format